Algorithmica (2007) 49: 13–36
DOI: 10.1007/s00453-007-0029-z

Algorithmica

The Complexity of the Single Individual SNP Haplotyping Problem[1]

Rudi Cilibrasi,[2] Leo van Iersel,[3] Steven Kelk,[2] and John Tromp[2]

Abstract. We present several new results pertaining to haplotyping. These results concern the combinatorial problem of reconstructing haplotypes from incomplete and/or imperfectly sequenced haplotype fragments. We consider the complexity of the problems *Minimum Error Correction* (MEC) and *Longest Haplotype Reconstruction* (LHR) for different restrictions on the input data. Specifically, we look at the *gapless* case, where every row of the input corresponds to a gapless haplotype-fragment, and the 1-*gap* case, where at most one gap per fragment is allowed. We prove that MEC is APX-hard in the 1-gap case and still NP-hard in the gapless case. In addition, we question earlier claims that MEC is NP-hard even when the input matrix is restricted to being completely binary. Concerning LHR, we show that this problem is NP-hard and APX-hard in the 1-gap case (and thus also in the general case), but is polynomial time solvable in the gapless case.

Key Words. Combinatorial algorithms, Complexity hierarchies, Complexity of approximation, Computational biology, Genetics.

1. Introduction. If we abstractly consider the human genome as a string over the nucleotide alphabet $\{A, C, G, T\}$, it is widely known that the genomes of any two humans have the same nucleotide at more than 99% of the sites. The sites at which variability is observed across the human population are called *Single Nucleotide Polymorphisms* (SNPs), which are formally defined as the sites on the human genome where, across the human population, two or more nucleotides are observed and each such nucleotide occurs in at least 5% of the population. These sites, which occur (on average) approximately once per thousand bases, capture the bulk of human genetic variability; the string of nucleotides found at the SNP sites of a human—the *haplotype* of that individual—can thus be thought of as a "fingerprint" for that individual.

It has been observed that, for most SNP sites, only two nucleotides are seen; sites where three or four nucleotides are found are comparatively rare. Thus, from a combinatorial perspective, a haplotype can be abstractly expressed as a string over the alphabet $\{0, 1\}$. Indeed, the biologically motivated field of SNP and haplotype analysis has spawned a rich variety of combinatorial problems, which are well described in surveys such as [1] and [2].

[1] Part of this research has been funded by the Dutch BSIK/BRICKS project. The first author was supported in part by NWO Project 612.55.002, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.
[2] Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. {Rudi.Cilibrasi,S.M.Kelk,John.Tromp}@cwi.nl.
[3] Technische Universiteit Eindhoven, Den Dolech 2, 5612 AX Eindhoven, The Netherlands. l.j.j.v.iersel@tue.nl.

Received October 25, 2005; revised March 3, 2006. Communicated by T. Jiang.
Online publication August 3, 2007.

We focus on two such combinatorial problems, both variants of the *Single Individual Haplotyping Problem* (SIH), introduced in [3]. SIH amounts to determining the haplotype of an individual using (potentially) incomplete and/or imperfect fragments of sequencing data. The situation is further complicated by the fact that, being a *diploid* organism, a human has two versions of each chromosome; one each from the individual's mother and father. Hence, for a given interval of the genome, a human has two haplotypes. Thus, SIH can be more accurately described as finding the two haplotypes of an individual given fragments of sequencing data where the fragments potentially have read errors and, crucially, where it is *not* known which of the two chromosomes each fragment was read from. We consider two well-known variants of the problem: *Minimum Error Correction* (MEC) and *Longest Haplotype Reconstruction* (LHR).

The input to these problems is a matrix M of haplotype fragments. Each column of M represents an SNP site and thus each entry of the matrix denotes the (binary) choice of nucleotide seen at that SNP location on that fragment. An entry of the matrix can thus either be "0", "1" or a *hole*, represented by "$-$", which denotes lack of knowledge or uncertainty about the nucleotide at that site. We use $M[i, j]$ to refer to the value found at row i, column j of M, and use $M[i]$ to refer to the ith row. Two rows r_1, r_2 of the matrix *conflict* if there exists a column j such that $M[r_1, j] \neq M[r_2, j]$ and $M[r_1, j], M[r_2, j] \in \{0, 1\}$.

A matrix is *feasible* iff the rows of the matrix can be partitioned into two sets such that all rows within each set are pairwise non-conflicting.

The objective in MEC is to "correct" (or "flip") as few entries of the input matrix as possible (i.e. convert 0 to 1 or vice versa) to arrive at a feasible matrix. The motivation behind this is that all rows of the input matrix were sequenced from one haplotype or the other, and that any deviation from that haplotype occurred because of read-errors during sequencing.

Problem LHR has the same input as MEC but a different objective. Recall that the rows of a feasible matrix M can be partitioned into two sets such that all rows within each set are pairwise non-conflicting. Having obtained such a partition, we can reconstruct a haplotype from each set by merging all the rows in that set together. (We define this formally later in Section 3.) With LHR the objective is to remove *rows* such that the resulting matrix is feasible and such that the sum of the lengths of the two resulting haplotypes is maximised.

In the context of haplotyping, MEC and LHR have been discussed—sometimes under different names—in papers such as [1], [4], [5] and (implicitly) [3]. One question arising from this discussion is how the distribution of holes in the input data affects computational complexity. To explain, we first define a *gap* (in a string over the alphabet $\{0, 1, -\}$) as a maximal contiguous block of holes that is flanked on both sides by non-hole values. For example, the string `---0010---` has no gaps `-0--10-111` has two gaps and `-0-----1--` has one gap. Two special cases of MEC and LHR that are considered to be practically relevant are the gapless case and the 1-gap case. The gapless variant is where every row of the input matrix is gapless, i.e. all holes appear at the start or end. In the 1-gap case every row has at most one gap.

In Section 2.1 we offer what we believe is the first proof that Gapless-MEC (and hence 1-gap MEC and also the general MEC) is NP-hard. We do so by reduction from MAX-CUT. (As far as we are aware, other claims of this result are based explicitly or

implicitly on results found in [6]; as we discuss in Section 2.3, we conclude that the results in [6] cannot be used for this purpose.)

The NP-hardness of 1-gap MEC (and general MEC) follows immediately from the proof that Gapless-MEC is NP-hard. However, our NP-hardness proof for Gapless-MEC is not approximation-preserving, and consequently tells us little about the (in)approximability of Gapless-MEC, 1-gap MEC and general MEC. In light of this we provide (in Section 2.2) a proof that 1-gap MEC is APX-hard, thus excluding (unless $P = NP$) the existence of a *Polynomial Time Approximation Scheme* (PTAS) for 1-gap MEC (and general MEC.)

We define (in Section 2.3) the problem *Binary-MEC*, where the input matrix contains no holes; as far as we know the complexity of this problem is still—intriguingly—open. In Section 2.4 we prove an "auxiliary" lemma which, besides being interesting in its own right, takes on a new significance in light of the open complexity of Binary-MEC. Subsequently, we consider a parameterised version of binary-MEC, where the number of haplotypes is not fixed as two, but is part of the input. We prove that this problem is NP-hard in Section 2.5.

In Section 3.1 we show that *Gapless-LHR* is polynomial-time solvable and give a dynamic programming algorithm for this which runs in time $O(n^2m + n^3)$ for an $n \times m$ input matrix. This improves upon the result of [3] which also showed a polynomial-time algorithm for Gapless-LHR but under the restricting assumption of non-nested input rows.

We also prove, in Section 3.2, that LHR is APX-hard (and thus also NP-hard) in the general case, by proving the much stronger result that 1-gap LHR is APX-hard. Although there is a claim in [3], made very briefly, that LHR is NP-hard in general, this is not substantiated. Therefore, our result is the first proof of hardness for both 1-gap LHR and general LHR.

2. Minimum Error Correction (MEC). For a length-m string $X \in \{0, 1, -\}^m$, and a length-m string $Y \in \{0, 1\}^m$, we define $d(X, Y)$ as the number of *mismatches* between the strings, i.e. positions where X is 0 and Y is 1, or vice versa; holes do not contribute to the mismatch count. Recall the definition of *feasible* from earlier; an alternative, and equivalent, definition (which we use in the following proofs) is as follows. An $n \times m$ SNP matrix M is *feasible* iff there exist two strings (haplotypes) H_1, $H_2 \in \{0, 1\}^m$, such that for all rows r of M, $d(r, H_1) = 0$ or $d(r, H_2) = 0$.

Finally, a *flip* is where a 0 entry is converted to a 1, or vice versa. Flipping to or from holes is not allowed and the haplotypes H_1 and H_2 may not contain holes.

2.1. *Gapless-MEC*

Problem. *Gapless-MEC*
Input: A gapless SNP matrix M.
Output: The smallest number of flips needed to make M feasible.

LEMMA 1. *Gapless-MEC is NP-hard.*

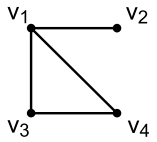

Fig. 2.1. Example input to MAX-CUT (see Lemma 1).

PROOF. We give a reduction from MAX-CUT, which is the problem of computing the size of a maximum cardinality cut in a graph. Let $G = (V, E)$ be the input to MAX-CUT, where E is undirected. (We identify, without loss of generality, V with $\{1, 2, \ldots, |V|\}$.) We construct an input matrix M for Gapless-MEC with $2k|V| + |E|$ rows and $2|V|$ columns where $k = 2|E||V|$. We use M_0 to refer to the first $k|V|$ rows of M, M_1 to refer to the second $k|V|$ rows of M and M_G to refer to the remaining $|E|$ rows. M_0 consists of $|V|$ consecutive blocks of k identical rows. Each row in the ith block (for $1 \leq i \leq |V|$) contains a 0 at columns $2i-1$ and $2i$ and holes at all other columns. M_1 is defined similar to M_0 with 1-entries instead of 0-entries. Each row of M_G encodes an edge from E: for edge $\{i, j\}$ (with $i < j$) we specify that columns $2i-1$ and $2i$ contain 0's, columns $2j-1$ and $2j$ contain 1's, and, for all $h \neq i, j$, column $2h-1$ contains 0 and column $2h$ contains 1. (See Figures 2.1 and 2.2 for an example of how M is constructed.)

Suppose t is the largest cut possible in G and s is the minimum number of flips needed to make M feasible. We claim that the following holds:

$$s = |E|(|V| - 2) + 2(|E| - t). \tag{2.1}$$

From this t, the optimal solution of MAX-CUT, can be easily computed. First, note that the solution to Gapless-MEC is trivially upperbounded by $|V||E|$. This follows because we could simply flip every 1 entry in M_G to 0; the resulting overall matrix would be feasible because we could just take H_1 as the all-0 string and H_2 as the all-1 string. Now, we say a haplotype H has the *double-entry* property if, for all odd-indexed positions (i.e. columns) j in H, the entry at position j of H is the same as the entry at position $j + 1$. We argue that a minimal number of feasibility-inducing flips will *always* lead to

$$\left.\begin{pmatrix}
0 & 0 & - & - & - & - & - & - \\
- & - & 0 & 0 & - & - & - & - \\
- & - & - & - & 0 & 0 & - & - \\
- & - & - & - & - & - & 0 & 0 \\
1 & 1 & - & - & - & - & - & - \\
- & - & 1 & 1 & - & - & - & - \\
- & - & - & - & 1 & 1 & - & - \\
- & - & - & - & - & - & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1
\end{pmatrix}\right\}\begin{array}{l}\left.\begin{array}{l}\\\\\\\\\\\\\\\\\end{array}\right\}\text{32 copies}\\\left.\begin{array}{l}\\\\\\\end{array}\right\}M_G\end{array}$$

Fig. 2.2. Construction of matrix M (from Lemma 1) for the graph in Figure 2.1.

two haplotypes H_1, H_2 such that both haplotypes have the double-entry property and, further, H_1 is the bitwise complement of H_2. (We describe such a pair of haplotypes as *partition-encoding*.) This is because if H_1, H_2 are not partition-encoding, then at least $k > |V||E|$ (in contrast with zero) entries in M_0 and/or M_1 will have to be flipped, meaning this strategy is doomed to begin with.

Now, for a given partition-encoding pair of haplotypes, it follows that—for each row in M_G—we will have to flip either $|V| - 2$ or $|V|$ entries to reach its nearest haplotype. This is because, irrespective of which haplotype we move a row to, the $|V| - 2$ pairs of columns *not* encoding endpoints (for a given row) will always cost one flip each to fix. Then either 2 or 0 of the four "endpoint-encoding" entries will also need to be flipped; four flips will never be necessary because then the row could move to the other haplotype, requiring no extra flips. Gapless-MEC thus maximises the number of rows which require $|V| - 2$ rather than $|V|$ flips. If we think of H_1 and H_2 as encoding a partition of the vertices of V (i.e. a vertex i is on one side of the partition if H_1 has 1's in columns $2i - 1$ and $2i$, and on the other side if H_2 has 1's in those columns), it follows that each row requiring $|V| - 2$ flips corresponds to a cut-edge in the vertex partition defined by H_1 and H_2. Expression (2.1) follows. □

2.2. 1-*Gap MEC*

Problem. 1-*gap MEC*
Input: SNP matrix M with at most one gap per row.
Output: The smallest number of flips needed to make M feasible.

To prove that 1-gap MEC is APX-hard we describe an *L-reduction*. This is a specific type of *approximation-preserving* reduction, first introduced in [7]. If there exists an L-reduction from a problem X to a problem Y, then a PTAS for Y can be used to build a PTAS for X. Conversely, if there exists an L-reduction from X to Y, and X is APX-hard, so is Y. See (for example) [8] for a succinct discussion of this. We will reduce from CUBIC-MIN-UNCUT, which is the problem of finding the minimum number of edges that have to be removed from a 3-regular graph in order to make it bipartite. Our first goal is thus to prove the APX-hardness of CUBIC-MIN-UNCUT, which itself will be proven using an L-reduction from the APX-hard problem CUBIC-MAX-CUT.

To aid the reader, we reproduce here the definition of an L-reduction.

DEFINITION 1 [7]. Let A and B be two optimisation problems. An *L-reduction* from A to B is a pair of functions R and S, both computable in polynomial time, such that for any instance I of A with optimum cost $\mathrm{Opt}(I)$, $R(I)$ is an instance of B with optimum cost $\mathrm{Opt}(R(I))$ and for every feasible solution s of $R(I)$, $S(s)$ is a feasible solution of I such that

$$\mathrm{Opt}(R(I)) \leq \alpha \mathrm{Opt}(I), \tag{2.2}$$

for some positive constant α and

$$|\mathrm{Opt}(I) - c(S(s))| \leq \beta |\mathrm{Opt}(R(I)) - c(s)|, \tag{2.3}$$

for some positive constant β, where $c(S(s))$ and $c(s)$ represent the costs of $S(s)$ and s, respectively.

OBSERVATION 1. *CUBIC-MIN-UNCUT is APX-hard.*

PROOF. We give an L-reduction from CUBIC-MAX-CUT, the problem of finding the maximum cardinality of a cut in a 3-regular graph. (This problem is shown to be APX-hard in [9]; see also [10].) Let $G = (V, E)$ be the input to CUBIC-MAX-CUT.

Note that CUBIC-MIN-UNCUT is the "complement" of CUBIC-MAX-CUT, as expressed by the following relationship:

$$\text{CUBIC-MAX-CUT(G)} = |E| - \text{CUBIC-MIN-UNCUT(G)}. \tag{2.4}$$

Here, and throughout this paper, we use P(I) to denote the optimal value of problem P on input I.

To see why (2.4) holds, note that for every cut C, the removal of the edges in $E\backslash C$ will lead to a bipartite graph. On the other hand, given a set of edges E' whose removal makes G bipartite, the complement is not necessarily a cut. However, given a bipartition induced by the removal of E', the edges from the original graph that cross this bipartition form a cut C', such that $|C'| \geq |E\backslash E'|$. This proves (2.4), and the mapping (just described) from E' to C' is the mapping we use in the L-reduction.

Now, note that property (2.2) of the L-reduction is easily satisfied (taking $\alpha = 1$) because the optimal value of CUBIC-MIN-UNCUT is always less than or equal to the optimal value of CUBIC-MAX-CUT. This follows from the combination of (2.4) with the fact that a maximum cut in a 3-regular graph always contains at least two-thirds of the edges: if a vertex has less than two incident edges in the cut then we can get a larger cut by moving this vertex to the other side of the partition.

To see that property (2.3) of the L-reduction is easily satisfied (taking $\beta = 1$), let E' be any set of edges whose removal makes G bipartite. Property (2.3) is satisfied because E' gets mapped to a cut C', as defined above, and combined with (2.4) this gives

$$\begin{aligned} \text{CUBIC-MAX-CUT(G)} - |C'| &\leq \text{CUBIC-MAX-CUT(G)} - |E\backslash E'| \\ &= |E'| - \text{CUBIC-MIN-UNCUT(G)}. \end{aligned} \tag{2.5}$$

This completes the L-reduction from CUBIC-MAX-CUT to CUBIC-MIN-UNCUT, proving the APX-hardness of CUBIC-MIN-UNCUT. □

We also need the following observation.

OBSERVATION 2. *Let $G = (V, E)$ be an undirected, 3-regular graph. Then we can find, in polynomial time, an orientation of the edges of G so that each vertex has either in-degree 2 and out-degree 1 ("in-in-out") or out-degree 2 and in-degree 1 ("out-out-in").*

PROOF. (We assume that G is connected; if G is not connected, we can apply the following argument to each component of G in turn, and the overall result still holds.) Every cubic graph has an even number of vertices, because every graph must have an

even number of odd-degree vertices. We add an arbitrary perfect matching to the graph, which may create multiple edges. The graph is now 4-regular and therefore has an Euler tour. We direct the edges following the Euler-tour; every vertex is now in-in-out-out. If we remove the perfect matching edges we added, we are left with an oriented version of G where every vertex is in-in-out or out-out-in. This can all be done in polynomial time. □

LEMMA 2. 1-*gap MEC is APX-hard*.

PROOF. We give a reduction from CUBIC-MIN-UNCUT. Consider an arbitrary 3-regular graph $G = (V, E)$ and orient the edges as described in Observation 2 to obtain an oriented version of G, $\overrightarrow{G} = (V, \overrightarrow{E})$, where each vertex is either in-in-out or out-out-in. We construct an $|E| \times |V|$ input matrix M for 1-gap MEC as follows. The columns of M correspond to the vertices of $\overrightarrow{G}$ and every row of M encodes an oriented edge of $\overrightarrow{G}$; it has a 0 in the column corresponding to the tail of the edge (i.e. the vertex from which the edge leaves), a 1 in the column corresponding to the head of the edge and it has holes in the remaining columns.

We prove the following:

$$\text{CUBIC-MIN-UNCUT(G)} = \text{1-gap MEC(M)}. \tag{2.6}$$

We first prove that

$$\text{1-gap MEC(M)} \leq \text{CUBIC-MIN-UNCUT(G)}. \tag{2.7}$$

To see this, let E' be a minimal set of edges whose removal makes G bipartite, and let $|E'| = k$. Let $B = (L \cup R, E \backslash E')$ be the bipartite graph (with bipartition $L \cup R$) obtained from G by removing the edges E'. Let H_1 (respectively, H_2) be the haplotype that has 1's in the columns representing vertices of L (respectively, R) and 0's elsewhere. It is possible to make M feasible with k flips, by the following process: for each edge in E', flip the 0 bit in the corresponding row of M to 1. For each row r of M it is now true that $d(r, H_1) = 0$ or $d(r, H_2) = 0$, proving the feasibility of M.

The proof that

$$\text{CUBIC-MIN-UNCUT(G)} \leq \text{1-gap MEC(M)} \tag{2.8}$$

is more subtle. Suppose we can render M feasible using j flips, and let H_1 and H_2 be any two haplotypes such that, after the j flips, each row of M is distance 0 from either H_1 or H_2. If H_1 and H_2 are bitwise complementary then we can make G bipartite by removing an edge whenever we had to flip a bit in the corresponding row. The idea is, namely, that the 1's in H_1 (respectively, H_2) represent the vertices L (respectively, R) in the resulting bipartition $L \cup R$.

However, suppose the two haplotypes H_1 and H_2 are not bitwise complementary. In this case it is sufficient to demonstrate that there also exists bitwise complementary haplotypes H_1' and H_2' such that, after j (or fewer) flips, every row of M is distance 0 from either H_1' or H_2'. Consider thus a column of H_1 and H_2 where the two haplotypes

are not complementary. Crucially, the orientation of $\overrightarrow{G}$ ensures that every column of M contains *either* one 1 and two 0's *or* two 1's and one 0 (and the rest holes). A simple case analysis shows that, because of this, we can always change the value of one of the haplotypes in that column, without increasing the number of flips. (The number of flips might decrease.) Repeating this process for all columns of H_1 and H_2 where the same value is observed thus creates complementary haplotypes H_1' and H_2', and—as described in the previous paragraph—these haplotypes then determine which edges of G should be removed to make G bipartite. This completes the proof of (2.6).

The above reduction can be computed in polynomial time and is an L-reduction. From (2.6) it follows directly that property (2.2) of an L-reduction is satisfied with $\alpha = 1$. Property (2.3), with $\beta = 1$, follows from the proof of (2.8), combined with (2.6). Namely, whenever we use (say) t flips to make M feasible, we can find $s \leq t$ edges of G that can be removed to make G bipartite. Combined with (2.6) this gives

$$|\text{CUBIC-MIN-UNCUT(G)} - s| \;\leq\; |\text{1-gap MEC(M)} - t|. \tag{2.9}$$ □

2.3. *Binary-MEC.* From a mathematical point of view it is interesting to determine whether MEC stays NP-hard when the input matrix is further restricted. We therefore define the following problem.

Problem Binary-MEC
Input: An SNP matrix M that does not contain any holes.
Output: The smallest number of flips needed to make M feasible.

Like all optimisation problems, the problem Binary-MEC has different variants. The above definition is technically speaking the *evaluation* variant of the Binary-MEC problem. See [11] for a more detailed explanation of terminology in this area. We now consider the *constructive* version:

Problem. Binary-Constructive-MEC
Input: An SNP matrix M of size $n \times m$ that does not contain any holes.
Output: Two haplotypes $H_1, H_2 \in \{0, 1\}^m$ minimizing

$$D_M(H_1, H_2) = \sum_{\text{rows } r \text{ of } M} \min(d(r, H_1), d(r, H_2)). \tag{2.10}$$

In the next subsection we prove that Binary-Constructive-MEC is polynomial-time Turing interreducible with its evaluation counterpart, Binary-MEC. This proves that Binary-Constructive-MEC is solvable in polynomial time if and only if Binary-MEC is solvable in polynomial time. We mention this correspondence because, when expressed as a constructive problem, it can be seen that MEC is in fact a specific type of *clustering* problem, a topic of intensive study in the literature. More specifically, we are trying to find two representative "median" (or "consensus") strings such that the sum, over all input strings, of the distance between each input string and its nearest median, is minimised. This interreducibility is potentially useful because we now argue, in contrast to claims in the existing literature, that the complexity of Binary-MEC/Binary-Constructive-MEC is actually still open.

To elaborate, it is claimed in several papers (e.g. [12]) that a problem equivalent to Binary-Constructive-MEC is NP-hard. Such claims inevitably refer to the seminal paper *Segmentation Problems* by Kleinberg, Papadimitriou and Raghavan (KPR), which has appeared in multiple different forms since 1998 (e.g. [6], [13] and [14]). However, the KPR papers actually discuss two superficially similar, but essentially different, problems: one problem is essentially equivalent to Binary-Constructive-MEC, and the other is a more general (and thus, potentially, a more difficult) problem. This more general problem allows the entries of the input matrix to be drawn arbitrarily from $\mathbb{R}$, which makes it much easier to prove NP-hardness. Communication with the authors [15] has confirmed that they have no proof of hardness for the former problem, i.e. the problem that is essentially equivalent to Binary-Constructive-MEC.

Thus we conclude that the complexity of Binary-Constructive-MEC/Binary-MEC is still open. From an approximation viewpoint the problem has been quite well-studied; the problem has a *Polynomial Time Approximation Scheme* (PTAS) because it is a special form of the *Hamming* 2*-Median Clustering Problem*. A randomized PTAS was demonstrated in [16] and later a deterministic PTAS in [17]. Other approximation results appear in [6], [12], [14] and a heuristic for a similar problem appears in [4]. We also know that, if the number of haplotypes to be found is specified as part of the input (and not fixed as 2), the problem becomes NP-hard; we prove this in the following section. Finally, it may also be relevant that the "geometric" version of the problem (where rows of the input matrix are not drawn from $\{0, 1\}^m$ but from $\mathbb{R}^m$, and the Euclidean distance is used instead of the Hamming distance) is also open from a complexity viewpoint [16]. (However, the version using Euclidean-distance-squared *is* known to be NP-hard [18].)

2.4. *Interreducibility of MEC and Constructive-MEC*. The following lemma proves that MEC is solvable in polynomial time if and only if Constructive-MEC is solvable in polynomial time. The same holds for Binary-MEC and Binary-Constructive-MEC.

LEMMA 3. *MEC and Constructive-MEC are polynomial-time Turing interreducible*. (*Also*, *Binary-MEC and Binary-Constructive-MEC are polynomial-time Turing interreducible*.)

PROOF. We show interreducibility of MEC and Constructive-MEC in such a way that the interreducibility of Binary-MEC with Binary-Constructive-MEC also follows immediately from the reduction. This makes the reduction from Constructive-MEC to MEC quite complicated because we must thus avoid the use of holes.

1. Reducing MEC to Constructive-MEC is trivial because, given an optimal haplotype pair (H_1, H_2), $D_M(H_1, H_2)$ can be easily computed in polynomial time by summing $\min(d(H_1, r), d(H_2, r))$ over all rows r of the input matrix M.
2. Reducing Constructive-MEC to MEC is more involved. To prevent a particular special case which could complicate our reduction, we first check whether every row of M (i.e. the input to Constructive-MEC) is identical. If this is so, we can complete the reduction by simply returning (H_1, H_1) where H_1 is the first row of M. Hence, from this point onwards, we assume that M has at least two distinct rows.

Let $OptPairs(M)$ be the set of all unordered optimal haplotype pairs for M, i.e. the set of all (H_1, H_2) such that $D_M(H_1, H_2) = \text{MEC}(M)$. Given that all rows in M are not identical, we observe that there are no pairs of the form (H_1, H_1) in $OptPairs(M)$. This is because $D_M(H_1, H_1)$ is always larger than $D_M(H_1, r)$ for any row r in M that is not equal to H_1. Let $OptPairs(M, H') \subseteq OptPairs(M)$ be those elements $(H_1, H_2) \in OptPairs(M)$ such that $H_1 = H'$ or $H_2 = H'$. Let $g(r, H_1, H_2)$ be defined as $\min(d(r, H_1), d(r, H_2))$.

Consider the following two subroutines:

Subroutine: DFN ("Distance From Nearest Optimal Haplotype Pair")
Input: An $n \times m$ SNP matrix M and a vector $r \in \{0, 1\}^m$.
Output: The value d_{dfn} which we define as follows:

$$d_{\text{dfn}} = \min_{(H_1, H_2) \in OptPairs(M)} g(r, H_1, H_2).$$

Subroutine: ANCHORED-DFN ("Anchored Distance From Nearest Optimal Haplotype Pair")
Input: An $n \times m$ SNP matrix M, a vector $r \in \{0, 1\}^m$, and a haplotype H' such that $(H', H_2) \in OptPairs(M)$ for some H_2.
Output: The value d_{adfn}, defined as

$$d_{\text{adfn}} = \min_{(H_1, H_2) \in OptPairs(M, H')} g(r, H_1, H_2).$$

We assume the existence of implementations of DFN and ANCHORED-DFN which run in polynomial time whenever MEC runs in polynomial time. We use these two subroutines to reduce Constructive-MEC to MEC and then, to complete the proof, demonstrate and prove correctness of implementations for DFN and ANCHORED-DFN.

The general idea of the reduction from Constructive-MEC to MEC is to find some pair $(H_1, H_2) \in OptPairs(M)$ by first finding H_1 (using repeated calls to DFN) and then finding H_2 (by using repeated calls to ANCHORED-DFN with H_1 specified as the "anchoring" haplotype.) Throughout the reduction, the following two observations are important. Both follow immediately from the definition of D—i.e. (2.10).

OBSERVATION 3. *Let $M_1 \cup M_2$ be a partition of rows of the matrix M into two sets. Then, for all H_1 and H_2, $D_M(H_1, H_2) = D_{M_1}(H_1, H_2) + D_{M_2}(H_1, H_2)$.*

OBSERVATION 4. *Suppose an SNP matrix M_1 can be obtained from an SNP matrix M_2 by removing* 0 *or more rows from M_2. Then* $\text{MEC}(M_1) \leq \text{MEC}(M_2)$.

To begin the reduction, note that, for an arbitrary haplotype X, $\text{DFN}(M, X) = 0$ if and only if $(X, H_2) \in OptPairs(M)$ for some haplotype H_2. Our idea is thus that we initialise X to be all-0 and flip one entry of X at a time (i.e. change a 0 to a 1 or vice versa) until $\text{DFN}(M, X) = 0$; at that point $X = H_1$ (for some $(H_1, H_2) \in OptPairs(M)$.) Note that it is not possible that $\text{DFN}(M, X) = m$, because all $(H_1, H_2) \in OptPairs(M)$ are of the form $H_1 \neq H_2$, and if $H_1 \neq H_2$ we know that $g(X, H_1, H_2) < m$. Suppose $\text{DFN}(M, X) = d$ where $0 < d < m$. If we define $flip(X, i)$ as the haplotype obtained by flipping the entry in the ith column of X, then we know that there exists i $(1 \leq i \leq m)$

such that DFN(M, *flip*(X, i)) $<$ d. Such a position must exist because we can flip some entry in X to bring it closer to the haplotype (which we know exists) that it was distance d from. It is clear that we can find a position i in polynomial time by calling DFN(M, *flip*(X, j)) for $1 \leq j \leq m$ until it is found. Having found such an i, we set $X = \mathit{flip}(X, i)$.

Clearly this process can be iterated, finding one entry to flip in every iteration, until DFN(M, X) $= 0$ and at this point setting $H_1 = X$ gives us the desired result. Given that DFN(M, X) decreases by at least 1 every iteration, at most $m - 1$ iterations are required.

Thus, having found H_1, we need to find some H_2 such that (H_1, H_2) is in *OptPairs*(M).

First, we initialise X to be the complement of H_1 (i.e. the row obtained by flipping every entry of H_1). Now, observe that if $X \neq H_1$ and ANCHORED-DFN(M, X, H_1) $= 0$ then (H_1, X) $\in$ *OptPairs*(M) and we are finished. The tactic is thus to find, at each iteration, some position i of X such that ANCHORED-DFN(M, *flip*(X, i), H_1) is less than ANCHORED-DFN(M, X, H_1), and then setting X to be *flip*(X, i). As before we repeat this process until our call to ANCHORED-DFN returns zero. The "trick" in this case is to prevent X converging on H_1, because (knowing that M has at least two different types of row) (H_1, H_1) $\notin$ *OptPairs*(M). The initialisation of X to the complement of H_1 guarantees this. To see why this is, observe that, if X is the complement of H_1, $d(X, H_1) = m$. Thus, we would need at least m flips to transform X into H_1. However, if X is the complement of H_1, then—because we have guaranteed that *OptPairs*(M) contains no pairs of the form (H_1, H_1)—we know that ANCHORED-DFN(M, X, H_1) $<$ m. Given that we can guarantee that ANCHORED-DFN(M, X, H_1) can be reduced by at least 1 at every iteration, it is clear that we can find an X such that ANCHORED-DFN(M, X, H_1) $= 0$ after making no more than $m - 1$ iterations, which ensures that X cannot have been transformed into H_1. Once we have such an X we can set $H_2 = X$ and return (H_1, H_2).

To complete the proof of Lemma 3 it remains only to demonstrate and prove the correctness of algorithms for DFN and ANCHORED-DFN, which we do below. Note that both DFN and ANCHORED-DFN run in polynomial time if MEC runs in polynomial time.

Subroutine: DFN ("Distance from Nearest Optimal Haplotype Pair")
Input: An $n \times m$ SNP matrix M and a vector $r \in \{0, 1\}^m$.
Output: The value d_{dfn} which we define as follows:

$$d_{\text{dfn}} = \min_{(H_1, H_2) \in \mathit{OptPairs}(M)} g(r, H_1, H_2).$$

The following is a three-step algorithm to compute DFN(M, r) which uses an oracle for MEC:

1. Compute $d = \text{MEC}(M)$.
2. Let M' be the $n(m + 1) \times m$ matrix obtained from M by making $m + 1$ copies of every row of M.
3. Return MEC($M' \cup \{r\}$) $- (m + 1)d$ where $M' \cup \{r\}$ is the matrix obtained by adding the single row r to the matrix M'.

To prove the correctness of the above we first make a further observation, which (as with the two previous observations) follows directly from (2.10).

OBSERVATION 5. *Suppose a $kn \times m$ SNP matrix M_1 is obtained from an $n \times m$ SNP matrix M_2 by making $k \geq 1$ copies of every row of M_2. Then* $\mathrm{MEC}(M_1) = k \cdot \mathrm{MEC}(M_2)$, *and* $\mathit{OptPairs}(M_1) = \mathit{OptPairs}(M_2)$.

By the above observation we know that $\mathrm{MEC}(M') = (m+1)d$ and $\mathrm{OptPairs}(M') = \mathit{OptPairs}(M)$. Now we argue that $\mathit{OptPairs}(M' \cup \{r\}) \subseteq \mathit{OptPairs}(M)$. To see why this is, suppose there existed (H_3, H_4) such that $(H_3, H_4) \in \mathit{OptPairs}(M' \cup \{r\})$ but $(H_3, H_4) \notin \mathit{OptPairs}(M)$. This would mean $D_M(H_3, H_4) > d$ where $d = \mathrm{MEC}(M)$. Now:

$$\begin{aligned} D_{M' \cup \{r\}}(H_3, H_4) &\geq D_{M'}(H_3, H_4) \\ &= (m+1) D_M(H_3, H_4) \\ &\geq (m+1)(d+1). \end{aligned}$$

However, if we take any $(H_1, H_2) \in \mathit{OptPairs}(M)$, we see that

$$\begin{aligned} D_{M' \cup \{r\}}(H_1, H_2) &\leq (m+1)d + g(r, H_1, H_2) \\ &\leq (m+1)d + m. \end{aligned}$$

Now $(m+1)d + m < (m+1)(d+1)$ so (H_3, H_4) could not possibly be in $\mathit{OptPairs}(M' \cup \{r\})$—contradiction! The relationship $\mathit{OptPairs}(M' \cup \{r\}) \subseteq \mathit{OptPairs}(M)$ thus follows. It further follows, from Observation 3, that the members of $\mathit{OptPairs}(M' \cup \{r\})$ are precisely those pairs $(H_1, H_2) \in \mathit{OptPairs}(M)$ that minimise the expression $g(r, H_1, H_2)$. The minimal value of $g(r, H_1, H_2)$ has already been defined as d_{dfn}, so we have

$$\mathrm{MEC}(M' \cup \{r\}) = (m+1)d + d_{\mathrm{dfn}}.$$

This proves the correctness of Step 3 of the subroutine.

Subroutine: ANCHORED-DFN ("Anchored Distance from Nearest Optimal Haplotype Pair")
Input: An $n \times m$ SNP matrix M, a vector $r \in \{0, 1\}^m$ and a haplotype H' such that $(H', H_2) \in \mathit{OptPairs}(M)$ for some H_2.
Output: The value d_{adfn}, defined as

$$d_{\mathrm{adfn}} = \min_{(H_1, H_2) \in \mathit{OptPairs}(M, H')} g(r, H_1, H_2).$$

Given that H' is one-half of some optimal haplotype pair for M, it can be shown that $\mathrm{ANCHORED\text{-}DFN}(M, r, H') = \mathrm{DFN}(M \cup \{H'\}, r)$, thus demonstrating how ANCHORED-DFN can be easily reduced to DFN in polynomial time. To prove the equation it is sufficient to demonstrate that $\mathit{OptPairs}(M \cup \{H'\}) = \mathit{OptPairs}(M, H')$, which we do now. Let $d = \mathrm{MEC}(M)$. It follows that $\mathrm{MEC}(M \cup \{H'\}) \geq d$. In fact, $\mathrm{MEC}(M \cup \{H'\}) = d$ because $D_{M \cup \{H'\}}(H', H_2) = d$ for all $(H', H_2) \in \mathit{OptPairs}(M, H')$. Hence

$OptPairs(M, H') \subseteq OptPairs(M \cup \{H'\})$. To prove the other direction, suppose there existed some pair $(H_1, H_2) \in OptPairs(M \cup \{H'\})$ such that $H_1 \neq H'$ and $H_2 \neq H'$. However, from Observation 3, we would then have

$$\begin{aligned} D_{M\cup\{H'\}}(H_1, H_2) &= D_M(H_1, H_2) + g(H', H_1, H_2) \\ &\geq D_M(H_1, H_2) + 1 \\ &> d. \end{aligned}$$

Thus, (H_1, H_2) could not have been in $OptPairs(M \cup \{H'\})$ in the first place, giving us a contradiction. Thus $OptPairs(M \cup \{H'\}) \subseteq OptPairs(M, H')$ and hence $OptPairs(M \cup \{H'\}) = OptPairs(M, H')$, proving the correctness of subroutine ANCHORED-DFN. □

2.5. *Parameterised Binary-MEC*. We now consider a generalisation of the problem Binary-MEC, where the number of haplotypes is not fixed as two, but as part of the input.

Problem: Parameterised-Binary-MEC (PBMEC)
Input: An SNP matrix M that contains no holes, and $k \in \mathbb{N}\backslash\{0\}$.
Output: The smallest number of flips needed to make M feasible under k haplotypes.

The notion of *feasible* generalises easily to $k \geq 1$ haplotypes: an SNP matrix M is *feasible* under k haplotypes if M can be partitioned into k segments such that all the rows within each segment are pairwise non-conflicting. The definition of D_M also generalises easily to k haplotypes; we define $D_{M,k}(H_1, H_2, \ldots, H_k)$ as

$$\sum_{\text{rows } r \text{ of } M} \min(d(r, H_1), d(r, H_2), \ldots, d(r, H_k)). \tag{2.11}$$

We define $OptTuples(M, k)$ as the set of unordered optimal k-tuples of haplotypes for M, i.e. those k-tuples of haplotypes which have a $D_{M,k}$ score equal to PBMEC(M, k).

LEMMA 4. *PBMEC is NP-hard.*

PROOF. We reduce from the NP-hard problem MINIMUM-VERTEX-COVER. Let $G = (V, E)$ be an undirected graph. A subset $V' \subseteq V$ is said to *cover* an edge $(u, v) \in E$ if $u \in V'$ or $v \in V'$. A *vertex cover* of an undirected graph $G = (V, E)$ is a subset U of the vertices such that every edge in E is covered by U. Given a graph G, MINIMUM-VERTEX-COVER is the problem of computing the size of a minimum cardinality vertex cover U of G.

Let $G = (V, E)$ be the input to MINIMUM-VERTEX-COVER. We construct an SNP matrix M as follows. M has $|V|$ columns and $3|E||V| + |E|$ rows. We name the first $3|E||V|$ rows M_0 and the remaining $|E|$ rows M_G. M_0 is the matrix obtained by taking the $|V| \times |V|$ identity matrix (i.e. 1's on the diagonal, 0's everywhere else) and making $3|E|$ copies of each row. Each row in M_G encodes an edge of G: the row has 1-entries at the endpoints of the edge, and the rest of the row is 0. For example if the

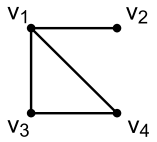

Fig. 2.3. Example input graph to MINIMUM-VERTEX-COVER (see Lemma 4).

input is the graph in Figure 2.3 then we construct the matrix from Figure 2.4. We argue shortly that to compute the size of the smallest vertex cover in G we call PBMEC(M, k) for increasing values of k (starting with $k = 2$) until we first encounter a k such that

$$\text{PBMEC}(M, k) = 3|E|(|V| - (k-1)) + |E|. \tag{2.12}$$

Once the smallest such k has been found, we can output that the size of the smallest vertex cover in G is $k - 1$. Actually, if we have not yet found a value $k < |V| - 2$ satisfying the above equation, we can check by brute force in polynomial time whether G has a vertex cover of size $|V| - 3$, $|V| - 2$, $|V| - 1$ or $|V|$. The reason for wanting to ensure that PBMEC(M, k) is not called with $k \geq |V| - 2$ is explained later in the analysis. Note that, should we wish to build a Karp reduction from the decision version of MINIMUM-VERTEX-COVER to the decision version of PBMEC, it is not a problem to make this brute-force checking fit into the framework of a Karp reduction. The Karp reduction can do the brute-force checking itself and use trivial inputs to the decision version of PBMEC to communicate its "yes" or "no" answer.

It remains only to prove that (for $k < |V| - 2$) (2.12) holds iff G has a vertex cover of size $k - 1$.

To prove this we first need to analyse $\mathit{OptTuples}(M_0, k)$. Recall that M_0 was obtained by duplicating the rows of the $|V| \times |V|$ identity matrix. Let $I_{|V|}$ be shorthand for the $|V| \times |V|$ identity matrix. Given that M_0 is simply a "scaled up" version of $I_{|V|}$, it follows that

$$\mathit{OptTuples}(M_0, k) = \mathit{OptTuples}(I_{|V|}, k). \tag{2.13}$$

Now, we argue that all the k-tuples in $\mathit{OptTuples}(I_{|V|}, k)$ (for $k < |V| - 2$) have the following form: one haplotype from the tuple contains only 0's, and the remaining $k - 1$ haplotypes from the tuple each have precisely one entry set to 1. We call such a k-tuple a *candidate* tuple.

$$\left.\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\right\} 12 \text{ copies}$$

$$\left.\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}\right\} M_G$$

Fig. 2.4. Construction of matrix M for graph from Figure 2.3.

First, note that $\text{PBMEC}(I_{|V|}, k) \leq |V| - (k-1)$, because $|V| - (k-1)$ is the value of the D measure—defined in (2.11)—under any candidate tuple. Secondly, under an arbitrary k-tuple there can be at most k rows of $I_{|V|}$ which contribute 0 to the D measure. However, if precisely k rows of $I_{|V|}$ contribute 0 to the D measure (i.e. every haplotype has precisely one entry set to 1, and the haplotypes are all distinct) then there are $|V| - k$ rows which each contribute 2 to the D measure; such a k-tuple cannot be optimal because it has a D measure of $2(|V| - k) > |V| - (k-1)$. So we reason that at most $k-1$ rows contribute 0 to the D measure. In fact, *precisely* $k-1$ rows must contribute 0 to the D measure because, otherwise, there would be at least $|V| - (k-2)$ rows contributing at least 1, and this is not possible because $\text{PBMEC}(I_{|V|}, k) \leq |V| - (k-1)$. So $k-1$ of the haplotypes correspond to rows of $I_{|V|}$, and the remaining $|V| - (k-1)$ rows of $I_{|V|}$ must each contribute 1 to the D measure. However, the only way to do this (given that $|V| - (k-1) > 2$) is to make the kth haplotype the haplotype where every entry is 0. Hence,

$$\text{PBMEC}(I_{|V|}, k) = |V| - (k-1) \tag{2.14}$$

and

$$\text{PBMEC}(M_0, k) = 3|E|(|V| - (k-1)). \tag{2.15}$$

$\mathit{OptTuples}(I_{|V|}, k)$ $(= \mathit{OptTuples}(M_0, k))$ is, by extension, precisely the set of candidate k-tuples.

The next step is to observe that $\mathit{OptTuples}(M, k) \subseteq \mathit{OptTuples}(M_0, k)$. To see this, suppose (by way of contradiction) that it is not true, and there exists a k-tuple $H^* \in \mathit{OptTuples}(M, k)$ that is not in $\mathit{OptTuples}(M_0, k)$. But then replacing H^* by any k-tuple out of $\mathit{OptTuples}(M_0, k)$ would reduce the number of flips needed in M_0 by at least $3|E|$, in contrast to an increase in the number of flips needed in M_G of at most $2|E|$, thus leading to an overall reduction in the number of flips; contradiction! (The $2|E|$ figure is the number of flips required to make all rows in M_G equal to the all-0 haplotype.)

Because $\mathit{OptTuples}(M, k) \subseteq \mathit{OptTuples}(M_0, k)$, we can restrict our attention to the k-tuples in $\mathit{OptTuples}(M_0, k)$. Observe that there is a natural 1-1 correspondence between the elements of $\mathit{OptTuples}(M_0, k)$ and all size $k-1$ subsets of V: a vertex $v \in V$ is in the subset corresponding to $H^* \in \mathit{OptTuples}(M_0, k)$ iff one of the haplotypes in H^* has a 1 in the column corresponding to vertex v.

Now, for a k-tuple $H^* \in \mathit{OptTuples}(M_0, k)$ we let $\mathit{Cov}(G, H^*)$ be the set of edges in G which are covered by the subset of V corresponding to H^*. (Thus, $|\mathit{Cov}(G, H^*)| = |E|$ iff H^* represents a vertex cover of G.) It is easy to check that, for $H^* \in \mathit{OptTuples}(M_0, k)$:

$$\begin{aligned} D_{M,k}(H^*) &= 3|E|(|V| - (k-1)) \\ &\quad + |\mathit{Cov}(G, H^*)| \\ &\quad + 2(|E| - |\mathit{Cov}(G, H^*|) \\ &= 3|E|(|V| - (k-1)) \\ &\quad + 2|E| - |\mathit{Cov}(G, H^*)|. \end{aligned}$$

Hence, for $H^* \in \mathit{OptTuples}(M_0, k)$, $D_{M,k}(H^*)$ equals $3|E|(|V| - (k-1)) + |E|$ iff H^* represents a size $k-1$ vertex cover of G. □

3. Longest Haplotype Reconstruction (LHR). Suppose a SNP matrix M is feasible. Then we can partition the rows of M into two sets, M_l and M_r, such that the rows within each set are pairwise non-conflicting. (The partition might not be unique.) From M_i ($i \in \{l, r\}$) we can then build a haplotype H_i by combining the rows of M_i as follows: The jth column of H_i is set to 1 if at least one row from M_i has a 1 in column j, is set to 0 if at least one row from M_i has a 0 in column j, and is set to a hole if all rows in M_i have a hole in column j. Note that, in contrast to MEC, this leads to haplotypes that potentially contain holes. For example, suppose one side of the partition contains rows `10--, -0--` and `---1`; then the haplotype we get from this is `10-1`. We define the *length* of a haplotype H, denoted as $|H|$, as the number of positions where it does not contain a hole; the haplotype `10-1` thus has length 3, for example. Now, the objective with LHR is to remove *rows* from M to make it feasible but also such that the sum of the lengths of the two resulting haplotypes is maximised. We define the function LHR(M) (which gives a natural number as output) as the largest value this sum-of-lengths value can take, ranging over all feasibility-inducing row-removals and subsequent partitions.

In Section 3.1 we provide a polynomial-time dynamic programming algorithm for the gapless variant of LHR, Gapless-LHR. In Section 3.2 we show that LHR becomes APX-hard and NP-hard when at most one gap per input row is allowed, automatically also proving the hardness of LHR in the general case.

3.1. *A Polynomial-Time Algorithm for Gapless-LHR*

Problem. Gapless-LHR
Input: A gapless SNP matrix M.
Output: The value LHR(M), as defined above.

The LHR problem for gapless matrices was proved to be polynomial-time solvable by Lancia et al. in [3], but only with the genuine restriction that no fragments are included in other fragments. Our algorithm improves this in the sense that it works for all gapless input matrices; our algorithm is similar in style to the algorithm by Bafna et al. [19] that solves MFR (minimum fragment removal), where the objective is not to maximise the length of the haplotypes, but to minimise the number of rows removed. Note that our dynamic-programming algorithm computes Gapless-LHR(M) but it can easily be adapted to generate the rows that must be removed (and subsequently, the partition that must be made) to achieve this value.

LEMMA 5. *Gapless-LHR can be solved in time* $O(n^2m + n^3)$.

PROOF. Let M be the input to Gapless-LHR, and assume the matrix has size $n \times m$. For row i define $l(i)$ as the leftmost column that is not a hole and define $r(i)$ as the rightmost column that is not a hole. The rows of M are ordered such that $l(i) \leq l(j)$ if $i < j$. Define the matrix M_i as the matrix consisting of the first i rows of M and two extra rows at the top: row 0 and row -1, both consisting of all holes. Define $W(i)$ as the set of rows $j < i$ that are not in conflict with row i.

For $h, k \leq i$ and $h, k \geq -1$ and $r(h) \leq r(k)$ define $D[h, k; i]$ as the maximum sum of lengths of two haplotypes such that:

- each haplotype is built up as a combination of rows from M_i (in the sense explained above);
- each row from M_i can be used to build at most one haplotype (i.e. it cannot be used for both haplotypes);
- row k is one of the rows used to build a haplotype and among such rows maximises $r(\cdot)$;
- row h is one of the rows used to build the haplotype for which k is not used and among such rows maximises $r(\cdot)$.

The optimal solution of the problem, LHR(M), is given by

$$\max_{h,k|r(h)\leq r(k)} D[h, k; n]. \tag{3.1}$$

This optimal solution can be calculated by starting with $D[h, k, 0] = 0$ for $h, k \in -1, 0$ and using the following recursive formulas. We distinguish three different cases, the first is that $h, k < i$. Under these circumstances,

$$D[h, k; i] = D[h, k; i - 1]. \tag{3.2}$$

This is because:

- if $r(i) > r(k)$: row i cannot be used for the haplotype that row k is used for, because row k has maximal $r(\cdot)$ among all rows that are used for a haplotype;
- if $r(i) \leq r(k)$: row i cannot increase the length of the haplotype that row k is used for (because also $l(i) \geq l(k)$);
- the same arguments hold for h.

The second case is when $h = i$, then $D[i, k; i]$ is equal to

$$\max_{\substack{j\in W(i),\ j\neq k \\ r(j)\leq r(i)}} D[j, k; i - 1] + f(i, j), \tag{3.3}$$

where $f(i, j) = r(i) - \max\{r(j), l(i) - 1\}$ is the increase of the haplotype's length. Equation (3.3) results from the following. The definition of $D[i, k; i]$ says that row i has to be used for the haplotype for which k is not used and amongst such rows maximises $r(\cdot)$. Therefore, the optimal solution is achieved by adding row i to some solution that has a row j as the most-right-ending row, for some j that agrees with i, is not equal to k and ends before i. Adding row i to the haplotype leads to an increase of its length of $f(i, j) = r(i) - \max\{r(j), l(i) - 1\}$. This term is fixed for fixed i and j and therefore we only have to consider extensions of solutions that were already optimal. Note that this reasoning does not hold for more general, "gapped", data.

The last case is when $k = i$, then $D[h, i; i]$ is equal to

$$\max_{\substack{j\in W(i),\ j\neq h \\ r(j)\leq r(i)}} \begin{cases} D[j, h; i - 1] + f(i, j) & \text{if} \quad r(h) \geq r(j), \\ D[h, j; i - 1] + f(i, j) & \text{if} \quad r(h) < r(j). \end{cases}$$

The above algorithm can be sped up by using the fact that, as a direct consequence of (3.2), $D[h, k; i] = D[h, k; \max(h, k)]$ for all $h, k \leq i \leq n$. It is thus unnecessary to calculate the values $D[h, k; i]$ for $h, k < i$.

The time for calculating all the $W(i)$ is $O(n^2m)$. When all the $W(i)$ are known, it takes $O(n^3)$ time to calculate all the $D[h, k; \max(h, k)]$. This is because we need to calculate $O(n^2)$ values $D[i, k; i]$ and also $O(n^2)$ values $D[h, i; i]$ that take $O(n)$ time each. This leads to an overall time complexity of $O(n^2m + n^3)$. □

3.2. 1-*gap LHR is NP-Hard and APX-Hard*

Problem. 1-gap LHR
Input: SNP matrix M with at most one gap per row.
Output: The value LHR(M), as defined earlier.

In this section we prove that 1-gap LHR is APX-hard (and thus also NP-hard.) We prove this by demonstrating (indirectly) an L-reduction from the problem CUBIC-MAX-INDEPENDENT-SET—the problem of computing the maximum cardinality of an independent set in a cubic graph—which is itself proven APX-hard in [9].

We reduce via the immediate problem *Single Haplotype LHR* (SH-LHR). In this version of the problem rows must be removed from the input matrix until the remaining rows are mutually non-conflicting. The objective is to maximise the number of columns that have at least one non-hole entry in the remaining rows.

The reduction chain looks as follows. We first show an L-reduction from SH-LHR to LHR, such that the number of gaps per row is unchanged. We then show an L-reduction from CUBIC-MAX-INDEPENDENT-SET to 2-gap SH-LHR. Next, using an observation pertaining to the structure of cubic graphs, we show how this reduction can be adapted to give an L-reduction from CUBIC-MAX-INDEPENDENT-SET to 1-gap SH-LHR. This proves the APX-hardness of 1-gap SH-LHR and thus (by transitivity of L-reductions) also 1-gap LHR.

LEMMA 6. *SH-LHR is L-reducible to LHR, such that the number of gaps per row is unchanged.*

PROOF. Let M be the $n \times m$ input to SH-LHR. We may assume that M contains no duplicate rows, because duplicate rows are redundant when working with only one haplotype. We map the SH-LHR input, M, to the $2n \times m$ LHR input, M', by taking each row of M and making a copy of it. Informally, the idea is that the influence of the second haplotype can be neutralised by doubling the rows of the input matrix. Note that this construction clearly preserves the maximum number of gaps per row.

Now, let $SOL(M')$ be the set that contains all pairs of haplotypes (H_1, H_2) that can be induced by removing some rows of M', partitioning the remaining rows of M' into two mutually non-conflicting sets, and then reading off the two induced haplotypes. Similarly, let $SOL(M)$ be the set that contains all haplotypes H that can be induced by removing some rows of M (such that the remaining rows are mutually non-conflicting) and then reading off the single, induced haplotype. Note the following pair of observations, which

both follow directly from the construction of M':

$$(H_1, H_2) \in SOL(M') \Rightarrow H_1, H_2 \in SOL(M), \tag{3.4}$$

$$H \in SOL(M) \Rightarrow (H, H) \in SOL(M'). \tag{3.5}$$

To satisfy the L-reduction we need to show how elements from $SOL(M')$ are mapped back to elements of $SOL(M)$ in polynomial time. So, let (H_1, H_2) be any pair from $SOL(M')$. If $|H_1| \geq |H_2|$ map the pair (H_1, H_2) to H_1, otherwise to H_2. This completes the L-reduction, and we now prove its correctness. Central to this is the proof of the following:

$$\text{SH-LHR}(M) = \tfrac{1}{2}\text{LHR}(M'). \tag{3.6}$$

The fact that $\text{SH-LHR}(M) \geq \frac{1}{2}\text{LHR}(M')$ follows immediately from (3.4) and the mapping described above. This lets us fulfil condition (2.2) of the L-reduction definition, taking $\alpha = 2$. The fact that $\text{SH-LHR}(M) \leq \frac{1}{2}\text{LHR}(M')$ follows because, by (3.5), every element in $SOL(M)$ is guaranteed to have a counterpart in $SOL(M')$ which has a total length twice as large.

We can fulfil condition (2.3) of the L-reduction by taking $\beta = \frac{1}{2}$. To see this, let (H_1, H_2) be any pair from $SOL(M')$, and (without loss of generality) assume that $|H_1| \geq |H_2|$. Let $r = \text{LHR}(M')$, the distance of (H_1, H_2) from optimal is then

$$r - (|H_1| + |H_2|) \geq r - 2|H_1|. \tag{3.7}$$

Let $l = \text{SH-LHR}(M)$, then

$$\begin{aligned} l - |H_1| &= \frac{r}{2} - |H_1| \\ &= \tfrac{1}{2}(r - 2|H_1|) \\ &\leq \tfrac{1}{2}(r - (|H_1| + |H_2|)). \end{aligned} \tag{3.8}$$

Thus, taking $\beta = \frac{1}{2}$ satisfies condition (2.3) of the L-reduction. □

LEMMA 7. *2-gap SH-LHR is APX-hard.*

PROOF. We reduce from CUBIC-MAX-INDEPENDENT-SET. Let $G = (V, E)$ be the undirected, cubic input to CUBIC-MAX-INDEPENDENT-SET. We direct the edges of G in the manner described by Observation 2, to give $\overrightarrow{G} = (V, \overrightarrow{E})$. Thus, every vertex of $\overrightarrow{G}$ is now out-out-in or in-in-out. A vertex w is a *child* of a vertex v if there is an edge leaving v in the direction of w, i.e. $(v, w) \in \overrightarrow{E}$, and in this case v is said to be the *parent* of w.

Let v_{in} be the number of vertices in $\overrightarrow{G}$ that are in-in-out, and let v_{out} be the number of vertices that are out-out-in. We build a matrix M, to be used as input to 2-gap SH-LHR, which has $|V|$ rows and $2v_{\text{in}} + v_{\text{out}}$ columns. The construction of M is as follows. (Each row of M will represent a vertex from V, so we henceforth index the rows of M using vertices of V.) Now, to each in-in-out vertex of $\overrightarrow{G}$, we allocate two *adjacent* columns of

M, and for each out-out-in vertex, we allocate one column of M. (A column may not be allocated to more than one vertex.) Note that, for this lemma, it is not important how the columns are allocated; in the proof of Lemma 10, the ordering is crucial. For simplicity, we also impose an arbitrary total order P on the vertices of V.

Now, for each vertex $v \in V$, we build row v as follows. Firstly, we put 1(s) in the column(s) representing v. Secondly, consider each child w of v. If w is an out-out-in vertex, we put a 0 in the column representing w. Alternatively, w is an in-in-out vertex, so w is represented by two columns; in this case we put a 0 in the left such column (if v comes before the other parent of w in the total order P) or, alternatively, in the right column (if v comes after the other parent of w in the total order P). The rest of the row consists of holes.

This completes the construction of M. Note that rows encoding in-in-out vertices contain two adjacent 1's and one 0, with at most one gap in the row, and rows encoding out-out-in vertices contain one 1 and two 0's, with at most two gaps in the row. In either case there are precisely three non-hole elements per row. It is also crucial to note that, reading down any one column of M, one sees exactly one 1 and exactly one 0.

Let K be any submatrix of M obtained by removing rows from M, and let $V[K] \subseteq V$ be the set of vertices whose rows appear in K. If the rows of K are mutually non-conflicting, then the haplotype induced by K has length $3r$ where r is the number of rows in K. This follows from the aforementioned facts that every column of M contains exactly one 1 and one 0, and that every row has exactly three non-hole elements.

We now prove that the rows of K are in conflict if and only if $V[K]$ is not an independent set. First, suppose $V[K]$ is not an independent set. Then there exist $u, v \in V[K]$ such that $(u, v) \in \overrightarrow{E}$. In row v of K there are thus 1's in the columns representing vertex v. However, there is also (in row u) a 0 in the column (or one of the columns) representing vertex v, causing a conflict. Hence, if $V[K]$ is not an independent set, K is in conflict. Now consider the other direction. Suppose K is in conflict. Then in some column of K there is a 0 and a 1. Let u be the row where the 0 is seen, and let v be the row where the 1 is seen. So both u and v are in $V[K]$. Further, we know that there is an out-edge (u, v) in $\overrightarrow{E}$, and thus an edge between u and v in E, proving that $V[K]$ is not an independent set. This completes the proof of the equivalence relationship.

It follows that

$$\text{CUBIC-MAX-INDEPENDENT-SET}(G) = \tfrac{1}{3}\text{SH-LHR}(M). \tag{3.9}$$

The conditions of the L-reduction definition are now easily satisfied, because of the one-to-one correspondence between haplotypes induced (after row-removals) and independent sets in G, and the fact that a size-r independent set of G corresponds to a length-$3r$ haplotype (or, equivalently, to r mutually non-conflicting rows of M.) The L-reduction is formally satisfied by taking $\alpha = 3$ and $\beta = \frac{1}{3}$. The two functions that comprise the L-reduction are both polynomial-time computable. □

LEMMA 8. *1-gap SH-LHR is APX-hard.*

PROOF. This proof is almost identical to the proof of Lemma 7; the difference is the manner in which columns of M are assigned to vertices of G. The informal motivation

is as follows. In the previous allocation of columns to vertices, it was possible for a row corresponding to an out-out-in vertex to have two gaps. Suppose, for each out-out-in vertex, we could ensure that one of the 0's in its row was adjacent to the 1 in the row, with no holes in between. Then every row of the matrix would have (at most) one gap, and we would be finished. We now show that, by exploiting a rather subtle property of cubic graphs, it is indeed possible to allocate columns to vertices such that this is possible.

Assume, that we have ordered the edges of G as before to obtain $\overrightarrow{G}$. Let $V_{\text{out}} \subseteq V$ be those vertices in V that are out-out-in. Now, suppose we could compute (in polynomial time) an injective function *favourite*: $V_{\text{out}} \to V$ with the following properties:

- for every $v \in V_{\text{out}}$, $(v, \mathit{favourite}(v)) \in \overrightarrow{E}$;
- the subgraph of $\overrightarrow{G}$ induced by edges of the form $(v, \mathit{favourite}(v))$, henceforth called the *favourite-induced subgraph*, is acyclic.

Given such a function it is easy to create a total enumeration of the vertices of V such that every out-out-in vertex is immediately followed by its *favourite* vertex. This enumeration can then be used to allocate the columns of M to the vertices of V, such that every row of M has at most one gap. To ensure this property, it is necessary to stipulate that, where $\mathit{favourite}(v)$ is an in-in-out vertex, the 0 encoding the edge $(v, \mathit{favourite}(v))$ is placed in the *left* of the two columns encoding $\mathit{favourite}(v)$. This is not a problem because every vertex is the favourite of at most one other vertex.

It remains to prove that the function *favourite* exists and that it can be constructed in polynomial time. This is equivalent to finding vertex disjoint directed paths in $\overrightarrow{G}$ such that every out-out-in vertex is on such a path and all paths end in an in-in-out vertex. Lemma 9 tells us how to find such paths. We thank Bert Gerards for invaluable help with this.

This completes the proof that 1-gap SH-LHR is APX-hard. (See Figures 3.1 and 3.2 for an example of the whole reduction in action.) □

LEMMA 9. *Let $\overrightarrow{G}$ be a directed, cubic graph with a partition $(V_{\text{out}}, V_{\text{in}})$ of the vertices such that the vertices in V_{out} are out-out-in and the vertices in V_{in} are in-in-out. Then V_{out} can be covered, in polynomial time, by vertex-disjoint directed paths ending in V_{in}.*

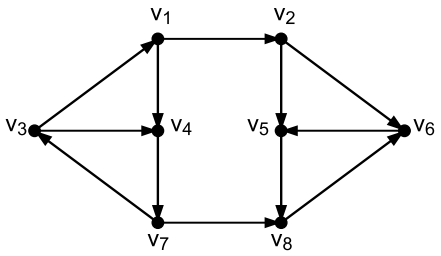

Fig. 3.1. Example input graph to CUBIC-MAX-INDEPENDENT-SET (see Lemmas 7 and 8) after an appropriate edge orientation has been applied.

$$
\begin{array}{c}
 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc} v_3 & v_1 & v_2 & v_5 & v_5 & v_7 & v_8 & v_8 & v_4 & v_4 & v_6 & v_6 \end{array} \\
\begin{pmatrix}
- & 1 & 0 & - & - & - & - & - & 0 & - & - & - \\
- & - & 1 & 0 & - & - & - & - & - & - & 0 & - \\
1 & 0 & - & - & - & - & - & - & - & 0 & - & - \\
- & - & - & - & - & 0 & - & - & 1 & 1 & - & - \\
- & - & - & 1 & 1 & - & - & 0 & - & - & - & - \\
- & - & - & - & 0 & - & - & - & - & - & 1 & 1 \\
0 & - & - & - & - & 1 & 0 & - & - & - & - & - \\
- & - & - & - & - & - & 1 & 1 & - & - & - & 0
\end{pmatrix}
\end{array}
$$

Fig. 3.2. Construction of matrix M (from Lemma 7 and 8) for graph in Figure 3.1.

PROOF. Observe that any two directed circuits contained entirely within V_{out} are pairwise vertex disjoint. Let V'_{out} be obtained from V_{out} by shrinking each directed circuit in V_{out} to a single vertex, and let $\overrightarrow{G'}$ be the resulting new graph. (Note that each vertex in V'_{out} has outdegree at least 2 and indegree at most 1 and that the indegree of each node in V_{in} is still 2, because we do not delete multiple edges.) We now argue that it is possible to find a set of edges F' in $\overrightarrow{G'}$, with $|F'| = |V'_{\text{out}}|$, such that, for each $v \in V'_{\text{out}}$, precisely one edge from F' begins at v, and such that no two edges in F' have the same endpoint. We prove this by construction. For each vertex $u \in V'_{\text{out}}$ that has a child v in V'_{out}, we can add the edge (u, v) to F', because v has indegree 1 and therefore no other edges can end at v. (In case u has two such children, we can choose one of the edges to add to F'.) Thus we are left to deal with a subset of vertices $L \subseteq V'_{\text{out}}$ where every vertex in L has all its children in V_{in}. Now consider the bipartite graph B with bipartition (L, V_{in}) and an edge for every directed edge of $\overrightarrow{G'}$ going from L to V_{in}. If we can find a matching in B of size $|L|$, we can complete the construction of F' by adding the edges from the perfect matching. Hall's theorem states that a bipartite graph with bipartition (X, Y) has a matching of size $|X|$ iff, for all $X' \subseteq X$, $|N(X')| \geq |X'|$, where $N(X')$ is the set of all neighbours of X'. Now, note that each vertex in L sends at least two edges across the partition of B, and each vertex in V_{in} can accept at most two such edges, so for each $L' \subseteq L$ it is clear that $|N(L')| \geq |L'|$. Hence, the graph (L, V_{in}) does indeed have a matching of size $|L|$ and the construction of F' can be completed.

Now, given that the graph induced by V'_{out} is acyclic, so is F'. Let F be the set of edges in $\overrightarrow{G}$ corresponding to those in F'. F is acyclic and each directed circuit C in V_{out} has exactly one vertex v_C that is a tail of an edge of F and no vertex that is a head of an edge in F. Let P_C be the longest directed path in C that ends in v_C. Then the union of F and all P_C over all directed circuits C in V_{out} is a collection of paths ending in V_{in} and covering V_{out}.

Finding cycles in a graph and finding a maximum matching in a bipartite graph are both polynomial-time computable, so the whole process described above is polynomial-time computable. □

LEMMA 10. 1-*gap LHR is APX-hard.*

PROOF. Follows from Lemmas 8 and 6. □

Table 1. The new state of knowledge following our work.

MEC	Binary (i.e. no holes)	? (Section 2.3)
		PTAS known [17]
	Gapless	NP-hard (Section 2.1)
	1-Gap	NP-hard (Section 2.2),
		APX-hard (Section 2.2)
LHR	Gapless	P (Section 3.1)
	1-Gap	NP-hard (Section 3.2)
		APX-hard (Section 3.2)
MFR	Gapless	P [19]
	1-Gap	NP-hard [3]
		APX-hard [19]
MSR	Gapless	P [3]
	1-Gap	NP-hard [19]
		APX-hard [19]

4. Conclusion. This paper involves the complexity (under various different input restrictions) of the haplotyping problems Minimum Error Correction (MEC) and Longest Haplotype Reconstruction (LHR). The state of knowledge about MEC and LHR after this paper is demonstrated in Table 1. We also include Minimum Fragment Removal (MFR) and Minimum SNP Removal (MSR) in the table because they are two other well-known Single Individual Haplotyping problems. MSR (MFR) is the problem of removing the minimum number of columns (rows) from an SNP matrix in order to make it feasible.

Indeed, from a complexity perspective, the most intriguing open problem is to ascertain the complexity of the "re-opened" problem Binary-MEC. It would also be interesting to study the approximability of Gapless-MEC.

From a more practical perspective, the next logical step is to study the complexity of these problems under more restricted classes of input, ideally under classes of input that have direct biological relevance. It would also be of interest to study some of these problems in a "weighted" context, i.e. where the cost of the operation in question (row removal, column removal, error correction) is some function of (for example) an *a priori* specified confidence in the correctness of the data being changed.

Acknowledgements. We thank Leen Stougie and Judith Keijsper for many useful conversations during the writing of this paper.

References

[1] Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Jing Li, The Haplotyping Problem: An Overview of Computational Models and Solutions, *Journal of Computer Science and Technology* **18**(6), 675–688 (2003).

[2] Bjarni V. Halldorsson, Vineet Bafna, Nathan Edwards, Ross Lippert, Shibu Yooseph, and Sorin Istrail, A Survey of Computational Methods for Determining Haplotypes, *Proceedings of the First RECOMB Satellite on Computational Methods for SNPs and Haplotype Inference*, Lecture Notes in Bioinformatics, 2983, pp. 26–47, Springer, New York, (2003).

[3] Giuseppe Lancia, Vineet Bafna, Sorin Istrail, Ross Lippert, and Russel Schwartz, SNPs Problems, Complexity and Algorithms, *Proceedings of the* 9*th Annual European Symposium on Algorithms*, pp. 182–193 (2001).

[4] Alessandro Panconesi and Mauro Sozio, Fast Hare: A Fast Heuristic for Single Individual SNP Haplotype Reconstruction, *Proceedings of* 4*th Workshop on Algorithms in Bioinformatics* (*WABI* 2004), LNB1 3240, pp. 266–277, Springer-Verlag, Berlin (2005).

[5] Harvey J. Greenberg, William E. Hart, and Giuseppe Lancia, Opportunities for Combinatorial Optimisation in Computational Biology, *INFORMS Journal on Computing*, **16**(3), 211–231 (2004).

[6] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan, Segmentation Problems, *Proceedings of STOC* 1998, pp. 473–482 (1998).

[7] Christos Papadimitriou and Mihelis Yannakakis, Optimization, Approximation, and Complexity Classes, *Journal of Computer and System Sciences* **43**, 425–440 (1991).

[8] Han Hoogeveen, Petra Schuurman, and Gerhard Woeginger, Non-Approximability Results for Scheduling Problems with Minsum Criteria, *INFORMS Journal on Computing*, **13**(2), 157–168 (2001).

[9] Paola Alimonti and Viggo Kann, Hardness of Approximating Problems on Cubic Graphs, *Proceedings of the Third Italian Conference on Algorithms and Complexity*, pp. 288–298 (1997).

[10] Piotr Berman and Marek Karpinski, On Some Tighter Inapproximability Results (Extended Abstract), *Proceedings of the* 26*th International Colloquium on Automata, Languages and Programming*, pp. 200–209 (1999).

[11] Giorgio Ausiello, Pilu Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela and Marco Protasi, *Complexity and Approximation—Combinatorial Optimization Problems and Their Approximability Properties*, Springer-Verlag, Berlin (1999).

[12] Noga Alon and Benny Sudakov, On Two Segmentation Problems, *Journal of Algorithms* **33**, 173–184 (1999).

[13] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan, A Microeconomic View of Data Mining, *Data Mining and Knowledge Discovery* **2**, 311–324 (1998).

[14] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan, Segmentation Problems, *Journal of the ACM* **51**(2), 263–280 (2004). Note: this paper is somewhat different to the 1998 version.

[15] Personal communication with Christos H. Papadimitriou, June 2005

[16] Rafail Ostrovsky and Yuval Rabani, Polynomial-Time Approximation Schemes for Geometric Min-Sum Median Clustering, *Journal of the ACM* **49**(2), 139–156 (March 2002).

[17] Yishan Jiao, Jingyi Xu, and Ming Li, On the k-Closest Substring and k-Consensus Pattern Problems, *Combinatorial Pattern Matching*: 15*th Annual Symposium* (*CPM* 2004), LNCS 3109, pp. 130–144, Springer-Verlag, Berlin (2004).

[18] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay, Clustering in Large Graphs via Singular Value Decomposition, *Journal of Machine Learning* **56**, 9–33 (2004).

[19] 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).