Acta Mathematicae Applicatae Sinica, English Series
Vol. 22, No. 3 (2006) 405–412

Acta Mathematicae Applicatae Sinica, English Series
© Springer-Verlag 2006

A Dynamic Programming Algorithm for the *k*-Haplotyping Problem

Zhen-ping Li[1,2], Ling-yun Wu[1], Yu-ying Zhao[3], Xiang-sun Zhang[1]

[1]Institute of Applied Mathematics, Academy of Mathematics and Systems Science, CAS, Beijing 100080, China (E-mail: wlyun@amt.ac.cn, zxs@amt.ac.cn)

[2]Mathematics Department of Beijing Wuzi University, Beijing, 101149, China. (E-mail: lizhenping@sina.com.cn)

[3] Mathematics Department of Beijing Forest University, Beijing, China. (E-mail: zhyuying@amss.ac.cn)

Abstract The Minimum Fragments Removal (MFR) problem is one of the haplotyping problems: given a set of fragments, remove the minimum number of fragments so that the resulting fragments can be partitioned into k classes of non-conflicting subsets. In this paper, we formulate the k-MFR problem as an integer linear programming problem, and develop a dynamic programming approach to solve the k-MFR problem for both the gapless and gap cases.

Keywords Integer programming, dynamic programming, k-haplotyping, SNP

2000 MR Subject Classification 90C39, 90C90, 92C40

1 Introduction

With complete genome sequences now available for humans and many other important organisms, it becomes a major challenge to locate genetic variants or polymorphism for prediction of disease using genomic data[2]. *Single nucleotide polymorphisms* (SNP) are the most frequent form of human genetic variants. A SNP is a single base pair position in genomic DNA where different nucleotide variants exist in some populations. We call each variant an allele. In a human body, SNPs are almost always *biallelic*, that is, there are two variants at each SNP site.

Diploid organisms, such as humans, possess two nearly identical copies of each chromosome[7], while multiploid organisms possess multiple nearly identical copies of each chromosome. A haplotype is a collection of SNPs on a single chromosome copy. A diploid individual SNPs can be combined into two haplotypes, while a multiploid individual SNPs can be combined into multiple haplotypes. In this paper, we will use number 1 and -1 to denote the two variants that each SNP can take. A haplotype is then a string over the number $\{1, -1\}$. Since DNA sequencing techniques such as Shotgun Sequencing[8] are restricted to small, overlapping fragments, which may contain errors (e.g. due to low quality reads) and can come from any one of the chromosome copies. The basic problem of k-haplotyping can be expressed as follows: *Given a set of fragments obtained by DNA sequencing from k ($k \geq 2$) copies of a chromosome, reconstruct k haplotypes from the SNPs value observed in the fragments.*

One of the most important k-haplotyping problem is minimum fragment removal (MFR) problem: *Given a set of fragments, remove the minimum number of "bad" fragments so that*

Manuscript received February 23, 2004. Revised February 14, 2006.
Supported by the National Natural Science Foundation of China (No. 60503004, No. 10471141).
The authors gratefully acknowledge the support of K.G.Wang Education Foundation of Hong Kong.

the resulting fragments can be divided into k disjoint sets of pairwise compatible fragments, each determining a haplotype. "Bad" fragments can be due to either contaminants (i.e. DNA coming from a different organism than the actual target) or read errors (i.e. a false 1, a false -1 inside a fragment). The problem was shown to be polynomial for gapless cases and NP-hard in general. The 2-MFR problem was well studied by G. Lancia et al.[3] and R. Rizzi et al.[5]. In [5], a practical algorithm was given for gapless or gap cases.

In this paper, we will generalize some results of [5] to the case of k-MFR problem. The rest of the paper is organized as follows. In next section, some basic terminologies and notations are introduced. Then an integer programming formulation for k-MFR problem is stated in Section 3. For the gapless case, a dynamic programming algorithm with polynomial time complexity is developed in Section 4 and the algorithm dealing with gaps is given in Section 5. The last section is the conclusions and discussions.

2 Terminology and Notation

Let $\mathcal{S} = \{1, 2, \cdots, n\}$ be the set of SNPs and $\mathcal{F} = \{1, 2, \cdots, m\}$ be the set of fragments. Each SNP is covered by some of the fragments, and can take two values, 1 or -1. Given an ordering of the SNPs, the data can be represented by an $m \times n$ matrix M over the number set $\{1, -1, 0\}$, which we call the SNP matrix. The number 0 is called a *hole*, which represents that a fragment does not cover a SNP site.

For two fragments f and g, we say that they are *conflict* if there is a SNP s, such that $M[f, s] \neq 0$, $M[g, s] \neq 0$ and $M[f, s] \neq M[g, s]$, where $M[g, s]$ is the element of M at row g and column s; otherwise, we say that they are *agree*. A SNP matrix M is called *feasible* if we can partition the rows (fragments) into k classes of non-conflicting fragments.

Given a SNP matrix M, the *fragment conflict graph* is a graph $G_{\mathcal{F}}(M) = (\mathcal{F}, E_{\mathcal{F}})$ with an edge for each pair of conflicting fragments, see Fig 1 for exemple.

$$\begin{pmatrix} 1 & -1 & 0 & 1 & 1 & 0 \\ -1 & 1 & 0 & 0 & -1 & 1 \\ 0 & 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & -1 & 0 & -1 & 1 \\ -1 & 0 & 1 & -1 & 1 & 0 \end{pmatrix}.$$

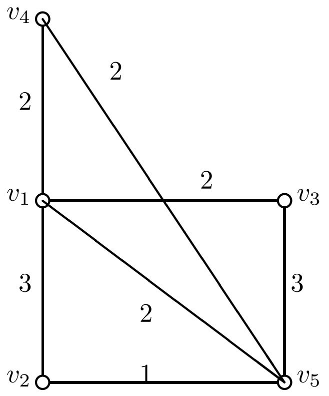

Figure 1. A SNP matrix and its fragment conflict graph

Note that if M is feasible, $G_{\mathcal{F}}(M)$ is $k-$colorable, since each haplotype defines an independent set of $G_{\mathcal{F}}(M)$, made of all the fragments coming from that haplotype. Conversely, if

$G_{\mathcal{F}}(M)$ is $k-$colorable, with color set $H_1, H_2, \cdots, H_k$, all the fragments in H_i $(1 \leq i \leq k)$ can be merged into one haplotype. Hence, M is feasible if and only if $G_{\mathcal{F}}(M)$ is $k-$colorable.

A *gapless fragment* is one covering a set of consecutive SNPs, that is, the 1s and (-1)s appear consecutively with no 0s between them. A *gap* is a maximal run of consecutive holes between two non-hole numbers. For example, 001-11-1-11000 is a gapless fragment, while there are two gaps in 001-1000100-1100. A fragment has r gaps if it covers $r+1$ blocks of consecutive SNPs. Such a fragment is equivalent to $r+1$ gapless fragments with the constraint that they must be put in the same haplotype or all discarded. The *length of a gap* is the number of holes it contains (e.g. 001-1000100-1100 has a total gap length of $5 = 3 + 2$). The *body* of a fragment extends from the leftmost non-hole to the rightmost non-hole (e.g. the body of 001-1-10100-1100 is 1-1-10100-11). In [3] the 2-MFR problem has been shown to be NP-hard in general, on the other hand, when all fragments are gapless, the 2-MFR problem can be solved in polynomial time.

In this paper, we will investigate the k-MFR problem and give the following results:

1. The k-MFR problem can be modelled as an integer linear programming problem.
2. The k-MFR problem is polynomial solvable in the gapless cases.
3. There is an $O(m^2n+m^{k+1})$ polynomial time algorithm for the k-MFR problem on matrices in which each fragment is gapless.
4. There is an $O(2^{2t}m^2n + 2^{(k+1)t}m^{k+1})$ polynomial time algorithm for the k-MFR problem on matrices in which each fragment has total gap length at most t.

3 Integer Programming Model

Denote by r_{ij} the jth SNP of the ith fragment, i.e. $M[i,j]$. Let x_{il} be a Boolean variable such that $x_{il} = 1$ if and only if fragment i belongs to haplotype l for $1 \leq l \leq k$. Let y_{lj} denote the jth SNP value of the lth haplotype for $1 \leq l \leq k$ and $1 \leq j \leq n$. Then the objective function of the k-MFR problem can be formulated as follows.

$$\max \sum_{i=1}^{m}\sum_{l=1}^{k} x_{il}. \tag{1}$$

The constraint set is as follows:

$$x_{il}(r_{ij} - y_{lj}) = 0, \qquad \begin{array}{l} i = 1, 2, \cdots, m; \\ j = 1, 2, \cdots, n; \\ l = 1, 2, \cdots, k; \\ r_{ij} \neq 0, \end{array} \tag{2}$$

$$\sum_{l=1}^{k} x_{il} \leq 1, \qquad i = 1, 2, \cdots, m, \tag{3}$$

$$x_{il} \in \{0, 1\}, \qquad i = 1, 2, \cdots, m; \quad l = 1, 2, \cdots k, \tag{4}$$

$$y_{lj} \in \{-1, 1\}, \qquad l = 1, 2, \cdots, k; \quad j = 1, 2, \cdots, n. \tag{5}$$

Constraint (2) says that if fragment i belongs to haplotype l, then they must agree. Constraint (3) guarantees that any fragment belongs to at most one haplotype. Constraint (4) and (5) are the integer constraints.

The non-linear constraints (2) can be replaced by the following linear inequations:

$$-2 + x_{il} \leq (r_{ij} - y_{lj}) \leq 2 - x_{il}, \qquad \begin{array}{l} i = 1, 2, \cdots, m; \\ j = 1, 2, \cdots, n; \\ l = 1, 2, \cdots, k, \\ r_{ij} \neq 0. \end{array} \tag{6}$$

Therefore, the k-MFR problem can be formulated as an integer linear programming with objective function (1) and constraint set (6), (3)–(5), thus solvable by a general integer linear programming method such as linear programming based branch and bound algorithm.

4 The Gapless Case

In this section, we will show that in the gapless case, the k-MFR problem can be solved in polynomial time, and a practical dynamic programming algorithm for the k-MFR problem is given. Throughout this section, assume that the SNP matrices are gapless.

4.1 k-MFR Problem Is Polynomial Solvable

Assume that there are no identical fragments, i.e., if two fragments are identical, they are denoted by one fragment. In this subsection, assume that there are no fragment inclusions or equals, i.e., denote by $l(i)$ and $r(i)$ the first and last SNP of a fragment i, $l(i) \leq l(j)$ implies $r(i) \leq r(j)$. We define a directed graph $D = (\mathcal{F}, A)$ as follows: Given two fragments i and j, with $l(i) \leq l(j)$, there is an arc $(i, j) \in A$ if i and j can be aligned without any mismatch, i.e., they agree in all their common SNPs (possibly none). Note that the common SNPs are a suffix of fragment i and a prefix of fragment j.

Lemma 1[5]. *Let M be a SNP matrix, $P_1, P_2, \cdots, P_k$ be node-disjoint directed paths in D such that $|V(P_1)| + |V(P_2)| + \cdots + |V(P_k)|$ is maximum. Let $\mathcal{R} = \mathcal{F} - \bigcup_{i=1}^{k} V(P_i)$, then $\mathcal{R}$ is a minimum set of fragments to remove such that $M[\mathcal{F} - \mathcal{R}]$ is feasible.*

Theorem 2. *There is a polynomial time algorithm for finding $P_1, P_2, \cdots, P_k$ in D such that $\sum_{i=1}^{k} |V(P_i)|$ is maximum.*

Proof. We can transform the problem into a maximum cost flow problem, which can be solved in polynomial time. We turn D into a network as follows. First, we introduce a dummy source s, a dummy sink t, and an arc (t, s) of capacity k and cost 0. s is connected to each node i with an arc (s, i) of cost 0, and each node i is connected to t, with cost 0 and capacity 1. Then we replace each node $i \in D$ with two nodes i' and i'' connected by an arc (i', i'') of cost 1 and capacity 1. All original arcs (u, v) of D are then replaced by arcs of type (u'', v'). A maximum cost circulation can be computed in polynomial time, by for example, linear programming[6]. Since D is acyclic, the solution is one cycle, which uses the arc (t, s) and then splits into k $s \to t$ directed paths, saturating as many arcs of type (i', i'') as possible of D. Since the capacity of arcs (i', i'') is 1, the paths are node-disjoint.

4.2 An $O(m^2n + m^{k+1})$ Dynamic Programming Algorithm

In what follows, we assume that one fragment can contain another one. In this subsection, we provide a dynamic programming approach for the solution of k-MFR. The resulting algorithm can be coded as to take $O(m^2n+m^{k+1})$ time. For convenience, in the following of this paper, we use $kMFR(M)$ to denote the optimal value of k-MFR problem when the SNP matrix is M.

Propositions 3 and 4 can be easily obtained by the similarity results of [5].

Proposition 3. *($\mathcal{S}$-reduction) Let M' be the matrix obtained from M by deleting columns where no 1 or no -1 occur. Clearly, $kMFR(M') \leq kMFR(M)$. Let $\mathcal{X}$ be any set of rows whose removal makes M' feasible, then $M \setminus \mathcal{X}$ is also feasible.*

Proposition 4. *($\mathcal{F}$-reduction) Let M' be the matrix obtained from M by deleting those rows conflicting with at most $k-1$ other rows. Clearly, $kMFR(M') \leq kMFR(M)$. Let $\mathcal{X}$ be any set of rows whose removal makes M' feasible, then $M \setminus \mathcal{X}$ is also feasible.*

We assume that the rows of M are ordered so that $l(i) \leq l(j)$ whenever $i < j$. For every $i \in \{1, 2, \cdots, m\}$, let M_i be the matrix made up by the first i rows of M. For $h_1, h_2, \cdots, h_k \leq i$ (with $h_j \geq -k+1$) such that $r(h_1) \leq r(h_2) \leq \cdots \leq r(h_k)$, we define $D[h_1, h_2, \cdots, h_k; i]$ as the minimum number of rows to remove to make M_i feasible, under the condition that

- Row h_k is not removed, and among the non-removed rows maximizes $r(h_k)$.
- Row h_j $(1 \leq j \leq k-1)$ is not removed and goes into another haplotype other than those of $h_{j+1}, \cdots, h_k$, and among such rows maximizes $r(h_j)$.

(If all rows of M_i are removed, then $h_1 = -k+1, h_2 = -k+2, \cdots, h_k = 0$ and $D[-k+1, -k+2, \cdots, 0; i] := i$, rows $-k+1, -k+2, \cdots, 0$ are all 0, that is, empty.)

Once all the $D[h_1, h_2, \cdots, h_k; i]$ are known, the solution to the k-MFR problem is given by

$$\min_{r(h_1)\leq r(h_2)\leq\cdots\leq r(h_k)} D[h_1, h_2, \cdots, h_k; m]. \tag{7}$$

Obviously, for every i, and for every $h_1, h_2, \cdots, h_k < i$ with $r(h_1) \leq r(h_2) \leq \cdots \leq r(h_k)$,

$$D[h_1, h_2, \cdots, h_k; i] := \begin{cases} D[h_1, h_2, \cdots, h_k; i-1], & \text{if } r(i) \leq r(h_j) \text{ for } 1 \leq j \leq k, \\ & \text{and rows } i \text{ and } h_j \text{ agree,} \\ D[h_1, h_2, \cdots, h_k; i-1] + 1, & \text{otherwise.} \end{cases} \tag{8}$$

Equation (8) can be proved by the following fact.

Lemma 5[5]**.** *Consider rows $a, b, c \in \mathcal{F}$. Assume $a, b < c$ and $r(a) \leq r(b)$. If a agrees with b and b agrees with c, then a agrees with c.*

For every i, we define $OK(i)$ as the set of those j with $j < i$ such that rows i and j agree. We assume $0, -1, \cdots, -k+1$, belong to $OK(i)$ for every i.

Now, for every i and every $h_1, h_2, \cdots, h_{k-1} < i$ with $r(h_1) \leq r(h_2) \leq \cdots \leq r(h_{k-1}) \leq r(i)$,

$$D[h_1, h_2, \cdots, h_{k-1}, i; i] := \min_{\substack{j\in OK(i)\\ j\neq h_1,h_2,\cdots,h_{k-1}\\ r(j)\leq r(i)}} \begin{cases} D[j, h_1, h_2, \cdots, h_{k-1}; i-1], & \text{if } r(h_1) \geq r(j), \\ D[h_1, h_2, \cdots, h_{k-1}, j; i-1], & \text{if } r(h_k) \leq r(j), \\ D[h_1, h_2, \cdots, h_l, j, h_{l+1}, \cdots, h_{k-1}; i-1], & \text{if } r(h_l) \leq r(j) \leq r(h_{l+1}). \end{cases} \tag{9}$$

Secondly, for every i and for every $h_1, h_2, \cdots, h_{k-1} < i$ with $r(i) \leq r(h_1) \leq r(h_2) \leq \cdots \leq r(h_{k-1})$,

$$D[i, h_1, h_2, \cdots, h_{k-1}; i] := \min_{j\in OK(i) j\neq h_1,h_2,\cdots,h_{k-1} r(j)\leq r(i)} D[j, h_1, h_2, \cdots, h_{k-1}; i-1]. \tag{10}$$

Finally, for every i, and for every $h_1, h_2, \cdots, h_{k-1} < i$ with $r(h_1) \le \cdots \le r(h_l) \le r(i) \le r(h_{l+1}) \le \cdots \le r(h_{k-1})$,

$$D[h_1, h_2, \cdots, h_l, i, h_{l+1}, \cdots, h_{k-1}; i] :=$$
$$\min_{\substack{j \in OK(i) \\ j \ne h_1, h_2, \cdots, h_{k-1} \\ r(j) \le r(i)}} \begin{cases} D[h_1, h_2, \cdots, h_l, j, h_{l+1}, \cdots, h_{k-1}; i-1], & \text{if } r(j) \ge r(h_l), \\ D[j, h_1, h_2, \cdots, h_{k-1}; i-1], & \text{if } r(j) \le r(h_1), \\ D[h_1, h_2, \cdots, h_p, j, h_{p+1}, \cdots, h_{k-1}; i-1], & \text{if } r(h_p) \le r(j) \le r(h_{p+1}). \end{cases} \quad (11)$$

Note that for computing the entries $D[h_1, h_2, \cdots, h_k; i]$, we only need to know the set $OK(i)$. The cost of creating the $OK(i)$ data structure is $O(m^2 n)$. The cost of computing the entries $D[h_1, h_2, \cdots, h_k; i]$ is $O(m^{k+1})$, since it can be seen as the cost of computing $O(m^{k+1})$ entries $D[h_1, h_2, \cdots, h_k; i]$ by using Equation (8) (costs $O(1)$ each) plus the cost of computing the $O(m^k)$ entries $D[h_1, \cdots, h_{k-1}, i; i]$, $D[i, h_1, \cdots, h_{k-1}; i]$ and $D[h_1, \cdots, h_l, i, h_{l+1}, \cdots, h_{k-1}; i]$ by using Equation (9)–(11) (costs $O(m)$ each).

5 Dealing With the Gaps

In this section, we propose a practical approach to deal with the k-MFR problem with gaps, when the number of holes in each fragment has an estimated upper bound. For the remainder of this section, let t be a constant such that the body of each fragment in the input instance contains at most t holes. We will derive a dynamic programming based polynomial algorithms. The resulting algorithm can be coded as to take $O(2^{2t} m^2 n + 2^{(k+1)t} m^{k+1})$ time.

Let f be a fragment and let $x \in \{1, -1\}^t$. We denote by $f[x]$ the fragment obtained from f by filling in the holes one by one, using the numbers in x. Since we assume that the body of each fragment in our input instance contains at most t holes, the numbers in x will always suffice to fill in all the holes of f.

Proposition 6. *Let*

$$\mathcal{F}_1 = \{f_1^1, f_1^2, \cdots, f_1^{m_1}\},\ \mathcal{F}_2 = \{f_2^1, f_2^2, \cdots, f_2^{m_2}\},\ \cdots,\ \mathcal{F}_k = \{f_k^1, f_k^2, \cdots, f_k^{m_k}\}$$

be sets of fragments in M such that any two fragments in $\mathcal{F}_i$ $(i = 1, 2, \cdots, k)$ agree. Then for every $p_i \le m_i$, $(i = 1, 2, \cdots, k)$, we can give $x_1^{p_1}, x_2^{p_2}, \cdots, x_k^{p_k} \in \{1, -1\}^t$ such that

$$\begin{aligned} \mathcal{F}_1' &= \{f_1^1, f_1^2, \cdots, f_1^{p_1}[x_1^{p_1}], \cdots, f_1^{m_1}\}, \\ \mathcal{F}_2' &= \{f_2^1, f_2^2, \cdots, f_2^{p_2}[x_2^{p_2}], \cdots, f_2^{m_2}\}, \\ &\vdots \\ \mathcal{F}_k' &= \{f_k^1, f_k^2, \cdots, f_k^{p_k}[x_k^{p_k}], \cdots, f_k^{m_k}\} \end{aligned}$$

would still be all without conflicts.

Assume that the rows of M are ordered so that $l(i) \le l(j)$ whenever $i < j$. For every $i \in \{1, 2, \cdots, m\}$, let M_i be the matrix made up by the first i rows of M. For $h_1, h_2, \cdots, h_k \le i$ (with $h_j \ge -k+1$, for $1 \le j \le k$) such that $r(h_1) \le r(h_2) \le \cdots \le r(h_k)$, and for $x^i \in \{1, -1\}^t$, we define $D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i]$ as the minimum number of rows to be removed to make $M_i[h_1[x^1], h_2[x^2], \cdots, h_k[x^k]]$ feasible, under the condition that

- Row $h_k[x^k]$ is not removed, and among the non-removed rows maximizes $r(h_k)$.
- Row $h_j[x^j]$ $(1 \le j \le k-1)$ is not removed and goes into another haplotype other than those of $h_{j+1}, \cdots, h_k$, and among such rows maximizes $r(h_j)$.

(If all rows are removed, then $h_1 = -k+1,\ h_2 = -k+2, \cdots,\ h_k = 0$, and $D[-k+1, x^1; -k+2, x^2; \cdots; 0, x^k; i] := i$ for all $x^i \in \{1,-1\}^t$).

Once all the $D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i]$ are known, the solution to the problem is given by

$$\min_{\substack{x^i \in \{1,-1\}^t, 1 \le i \le k \\ r(h_1) \le r(h_2) \le \cdots \le r(h_k)}} D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; m]. \tag{12}$$

Clearly, for every i, and for every $h_1, h_2, \cdots, h_k < i$ with $r(h_1) \le r(h_2) \le \cdots \le r(h_k)$,

$$D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i] := \begin{cases} D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i-1], & \text{if } r(i) \le r(h_k) \\ & \text{and rows } i \text{ and } h_k[x^k] \text{ agree,} \\ D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i-1], & \text{if } r(i) \le r(h_j) \text{ for } (1 \le j \le k-1) \\ & \text{and rows } i \text{ and } h_j[x^j] \text{ agree,} \\ D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i-1] + 1, & \text{otherwise.} \end{cases} \tag{13}$$

For every fragment i and for every $x \in \{1,-1\}^t$, we define $OK(i,x)$ as follows:

$$OK(i,x) = \{(j,y) | j < i,\ y \in \{1,-1\}^t,\ i[x] \text{ and } j[y] \text{ agree}\}. \tag{14}$$

Now, for every fragment i and for every $h_1, h_2, \cdots, h_k < i$ with $r(h_1) \le r(h_2) \le \cdots \le r(h_k) \le r(i)$, and for every $x^i \in \{1,-1\}^t$, $x^j \in \{1,-1\}^t$, $1 \le j \le k-1$,

$$D[h_1, x^1; h_2, x^2; \cdots; h_{k-1}, x^{k-1}; i, x^i; i] := \min_{\substack{(j,x^j) \in OK(i,x^i) \\ j \ne h_1, h_2, \cdots, h_{k-1} \\ r(j) \le r(i)}} \begin{cases} D[j, x^j; h_1, x^1; \cdots; h_{k-1}, x^{k-1}; i-1], & \text{if } r(j) \le r(h_1) \\ D[h_1, x^1; \cdots; h_{k-1}, x^{k-1}; j, x^j; i-1], & \text{if } r(h_k) \le r(j) \\ D[h_1, x^1; \cdots; h_l, x^l; j, x^j; h_{l+1}, x^{l+1}; \cdots; & \\ h_{k-1}, x^{k-1}; i-1], & \text{if } r(h_l) \le r(j) \le r(h_{l+1}). \end{cases} \tag{15}$$

Secondly, for every i and for every $h_1, h_2, \cdots, h_{k-1} < i$ with $r(i) \le r(h_1) \le r(h_2) \le \cdots \le r(h_{k-1})$ and for every $x^i \in \{1,-1\}^t$, $x^j \in \{1,-1\}^t$, $1 \le j \le k-1$,

$$D[i, x^i; h_1, x^1; \cdots; h_{k-1}, x^{k-1}; i] := \min_{\substack{(j,x^j) \in OK(i,x^i) \\ j \ne h_1, h_2, \cdots, h_{k-1} \\ r(j) \le r(i)}} D[j, x^j; h_1, x^1; \cdots; h_{k-1}, x^{k-1}; i-1]. \tag{16}$$

Finally, for every i and for every $h_1, h_2, \cdots, h_{k-1} < i$ with $r(h_1) \le r(h_2) \le \cdots \le r(h_l) \le r(i) \le r(h_{l+1}) \le \cdots \le r(h_{k-1})$ and for every $x^i \in \{1,-1\}^t$, $x^j \in \{1,-1\}^t$, $1 \le j \le k-1$,

$$D[h_1, x^1; h_2, x^2; \cdots; h_l, x^l; i, x^i; h_{l+1}, x^{l+1}; \cdots; h_{k-1}, x^{k-1}; i] := \min_{\substack{(j,x^j) \in OK(i,x^i) \\ j \ne h_1, h_2, \cdots, h_{k-1} \\ r(j) \le r(i)}} \begin{cases} D[h_1, x^1; \cdots; h_l, x^l; j, x^j; h_{l+1}, x^{l+1}; \cdots; h_{k-1}, x^{k-1}; i-1], & \text{if } r(j) \ge r(h_l) \\ D[j, x^j; h_1, x^1; \cdots; h_{k-1}, x^{k-1}; i-1], & \text{if } r(j) \le r(h_1) \\ D[h_1, x^1; \cdots; h_p, x^p; j, x^j; h_{p+1}, x^{p+1}; \cdots; h_{k-1}, x^{k-1}; i-1], & \\ & \text{if } r(h_p) \le r(j) \le r(h_{p+1}). \end{cases} \tag{17}$$

Note that for computing the entries $D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i]$, we only need to know the sets $OK(i)$, the cost of creating $OK(i, x^i)$ data structure (done in the first phase) is $O(2^{2t}m^2n)$, the cost of computing the entries $D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i]$ (done in a second phase) is $O(2^{(k+1)t}m^{k+1})$, since it can be seen as the cost of computing the $O(2^{(k+1)t}m^{k+1})$ entries $D[h_1, x^1; h_2, x^2; \cdots; h_k, x^k; i]$ with $h_1 < h_2 < \cdots < h_k < i$ by using Equation (13) plus the cost of computing the $O(2^{kt}m^k)$ entries $D[h_1, x^1; \cdots; i, x^i; \cdots; h_{k-1}, x^{k-1}; i]$ by using Equation (15)–(17) (cost $O(2^t m)$ each).

6 Conclusion

In this paper, we consider one of the namely the k-haplotyping problem, k-MFR problem, and give an integer programming model and a dynamic programming algorithm for both the gapless and gap cases. For m fragments and n SNPs, the complexity of the algorithm is $O(2^{2t}m^2n + 2^{(k+1)t}m^{k+1})$ when the body of each fragment has at most t holes. The future research work is to develop models and algorithms for other k-haplotyping problems such as k-MSR (minimum SNP removal) problem: Given a SNP matrix, remove the minimum number of SNPs (columns) such that the resulting fragments can be partitioned into k non-conflicting classes.

References

[1] Greenberg, H.J., Hart, W.E., Lancia, G. Opportunities for combinatorial optimization in computational Biology. *INFORMS Journal on Computing*, 16(3): 211–231 (2004)

[2] Lippert, R. Schwartza, R., Lancia, G., Istrail, S. Algorithmic strategies for the SNPs haplotype assembly problem. *Briefings in Bioinformatrics*, 3(1): 23–31 (2002)

[3] Lancia, G., Bafna, V., Istrail, S., Lippert, R., Schwartz, R. SNPs problems, complexity and algorithms. In Proceedings of Annual European Symposium on Algorithms (ESA), volume 2161, Lecture Notes in Computer Science, 182–193, Springer-Verlag, Berlin, Heidelberg, 2001

[4] Lund, C., Yannakakis, M. The approximation of maximum subgraph problems. In Proceedings of 20th Int. Colloqium on Automata, Languages and Programming, 40–51, Springer-Verlag, Berlin, Heidelberg, 1994

[5] Rizzi, R. Bafna, V., Istrail, S., Lancia, G. Practical algorithms and fixed-parameter tractability for the single individual SNP haploityping problem. In R. Guigo and D. Gusfield, editors. Proceedings of 2nd Annual Workshop on Algorithms in Bioinformaticcs (WABI), volum 2452 of Lecture Notes in Computer Science, 29–43, Springer-Verlag, Berlin, Heidelberg, 2002

[6] Tardos, E. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3): 247–255 (1985)

[7] Venter, J. et al. The sequence of the human genome. *Science*, 291: 1304–1351 (2001)

[8] Weber, J., Myers. E. Human whole genome shotgun sequencing. *Genome Research*, 7: 401–409 (1997)