Algorithmica
DOI 10.1007/s00453-012-9730-7

The Parameterized Complexity of the Shared Center Problem

Zhi-Zhong Chen · Wenji Ma · Lusheng Wang

Received: 8 February 2012 / Accepted: 10 December 2012

Abstract Recently, the *shared center* (SC) problem has been proposed as a mathematical model for inferring the allele-sharing status of a given set of individuals using a database of confirmed haplotypes as reference. The problem was proved to be NP-complete and a ratio-2 polynomial-time approximation algorithm was designed for its minimization version (called the *closest shared center* (CSC) problem). In this paper, we consider the parameterized complexity of the SC problem. First, we show that the SC problem is $W[1]$-hard with parameters d and n, where d and n are the *radius* and the number of (diseased or normal) individuals in the input, respectively. Then, we present two asymptotically optimal parameterized algorithms for the problem and apply them to linkage analysis.

Keywords Haplotype inference · Linkage analysis · Pedigree · Allele-sharing status · Parameterized complexity · Parameterized algorithms

1 Introduction

Linkage analysis is the first step to reduce the possible region for identifying a disease gene. Linkage studies have facilitated the identification of several hundred human

Z.-Z. Chen (✉)
Division of Information System Design, Tokyo Denki University, Ishizaka, Hatoyama, Hiki, Saitama 359-0394, Japan
e-mail: zzchen@mail.dendai.ac.jp

W. Ma · L. Wang
Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

W. Ma
e-mail: wenjima2@student.cityu.edu.hk

L. Wang
e-mail: lwang@cs.cityu.edu.hk

genes that can harbor mutations leading to a disease phenotype. The fundamental problem in linkage analysis is to identify regions whose allele is shared by all or most affected members but by none or few unaffected family members. Almost all the existing methods for linkage analysis are for families with clearly given pedigrees [1, 4, 9–11, 15, 16, 18, 19]. The pedigree information helps a lot for designing computational algorithms. Very few methods can handle the case when the sampled individuals are closely related but the real relationship is hidden (most of the times because of remote relationship). This situation occurs very often when the individuals share a common ancestor six or more generations ago.

With the new development of microarray techniques, high-density SNP genotype data can be used for large-scale and cost-effective linkage analysis. Recently, the international HapMap project has produced enormous amount of haplotype data for individuals in some major populations. For example, there are 340 haplotypes in the group "Japanese in Tokyo" + "Han Chinese in Beijing". These new developments make it possible to propose new mathematical models for finding genes causing genetic diseases when the sampled individuals are closely related but their pedigree is unknown.

The real problem is as follows: We are given three sets $D = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_k\}$, $N = \{\hat{g}_{k+1}, \ldots, \hat{g}_n\}$, and $H = \{\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_m\}$, where D consists of *diseased* individuals represented by their genotype data on a *whole* chromosome C, N consists of *normal* individuals represented by their genotype data on C, and H consists of confirmed haplotype data on C of some individuals in the same (or similar) population. For convenience, we call H the *reference database*. Note that H can be obtained from any haplotype database for a set of individuals, e.g., the database of HapMap project is available. A *region* on a chromosome, denoted by $[a, b]$, is a set of consecutive SNP sites (positions) starting at position a and ending at position b. The objective here is to find the *true mutation regions* of C. Here, a true mutation region of C means a consecutive portion of C where all the diseased individuals share a common haplotype segment that is shared by none of the normal individuals. The true mutation regions defined here are based on the haplotype segments of all individuals. If we know the haplotype segments of all the individuals, the true mutation regions can be easily computed. Thus, the challenge is to infer the haplotypes of each individual based on the input genotype data as well as the reference database H.

The first strike to the problem was given by Ma et al. [12]. In order to tackle the problem, Ma et al. proposed the following strategy: First, divide the whole chromosome into a set of (disjoint) regions of the same length L. Then, classify the length-L regions into *valid* or *invalid* regions based on a mathematical model (called the *shared center* (SC) problem). Finally, design a heuristic to merge/refine the valid regions to get predicted mutation regions. For details, see Ma et al. [12]. The key computational technique used in the above method is the proposed mathematical model (namely, the SC problem) for inferring the allele-sharing status of a given set of individuals using a database of confirmed haplotypes as reference.

We here only give a rough definition of the SC problem; the precise definition can be found in Sect. 3. An input to the SC problem is a quadruple (D, N, H, d), where D (respectively, N) consists of genotype segments of the same length L from diseased (respectively, normal) individuals, H is the reference database consisting of haplotype segments of length L, and d (referred to as the *radius*) is a nonnegative integer. The goal is to find a *center* haplotype segment s of length L and split each genotype segment $g_i \in D \cup N$ into a pair $(h_{i,1}, h_{i,2})$ of haplotype segments so that (1) each haplotype segment $h_{i,j}$ is within a Hamming distance of at most d from some segment in H, (2) $s = h_{i,1}$ for each $g_i \in D$ (i.e., the diseased individuals share the center haplotype segment), and (3) $h_{i,1} \neq s$ and $h_{i,2} \neq s$ for each $g_i \in N$ (i.e., the normal individuals do not share the center haplotype segment).

Ma et al. [12] show that the SC problem is NP-complete. They also consider the *closest shared center* (CSC) problem where the input is (D, N, H) and the goal is to find the minimum integer d such that the instance (D, N, H, d) of the SC problem has a solution. They propose a ratio-2 polynomial-time approximation algorithm for the CSC problem.

In this paper, we consider the parameterized complexity of the SC problem. First, we show that the SC problem is $W[1]$-hard with parameters d and n, where n is the number of (diseased or normal) individuals in the input. As a corollary of this result, we show that the SC problem on input of length $\tilde{n}$ cannot be solved in $O(f(d,n)\tilde{n}^{o(\log d)})$ time for any computable function f, as long as the following well-known conjecture (called the *Exponential Time Hypothesis* [8]) is true:

- *The ETH Conjecture:* There is no $O(2^{o(n)})$-time algorithm for deciding whether a given boolean formula $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ with n variables is satisfiable or not, where each C_i $(1 \le i \le m)$ is the disjunction of three literals.

We then present two parameterized algorithms for the SC problem. One of them runs in $O(m^3L(n-k) + m^2Lk + kd \cdot (6\sqrt{3})^d \cdot m^{\lfloor \log_2(d+1)\rfloor+2})$ time, where k is the number of diseased individuals and m is the number of haplotype segments in the reference database. The other takes $O(m^3L(n-k)+m^2Lk+k^2d \cdot 8^d \cdot m^{\lfloor \log_2(d+1)\rfloor+2})$ time. Note that the two algorithms are asymptotically optimal in the sense that the exponent of m in the running time cannot be improved to $o(\log d)$ as long as the ETH conjecture is true.

The remainder of this paper is organized as follows. Section 2 contains several basic definitions and notations. Section 3 contains the precise definition of the SC problem. In Sect. 4, we prove the hardness of the SC problem. Section 5 details two parameterized algorithms for the SC problem. Finally, we apply the algorithms to linkage analysis and compare their running time in Sect. 6.

2 Basic Definitions and Notations

For a finite set S, $|S|$ denotes the number of elements in S. Similarly, for a string s, $|s|$ denotes the length of s. A string s has $|s|$ positions, namely, $1, 2, \ldots, |s|$. For convenience, if L is a positive integer, then we use $[1..L]$ to denote the set $\{1, 2, \ldots, L\}$. The letter of s at position $i \in [1..|s|]$ is denoted by $s[i]$. Thus, $s = s[1]s[2]\cdots s[|s|]$. For two integers i and j with $1 \le i \le j \le |s|$, $s[i..j]$ denotes $s[i]s[i+1]\cdots s[j]$. For a binary string s, $\bar{s}$ denotes the *complement string* of s, where $\bar{s}[i] \neq s[i]$ for every $i \in [1..|s|]$. For two strings s and t of the same length, $\{s \not\equiv t\}$ denotes the set of all positions $i \in [1..|s|]$ with $s[i] \neq t[i]$, and $dist(s,t)$ denotes $|\{s \not\equiv t\}|$ (i.e., the Hamming distance between s and t). For a string s and a subset P of $[1..|s|]$, $s|_P$ denotes the string obtained from s by deleting all letters at positions not in P.

At last, when an algorithm exhaustively tries all possibilities to find the right choice, we say that the algorithm *guesses* the right choice.

3 The Shared Center Problem

An input to the SC problem is a quadruple (D, N, H, d), where $D = \{g_1, g_2, \dots, g_k\}$ and $N = \{g_{k+1}, g_{k+2}, \dots, g_n\}$ are sets consisting of genotype segments of the same length L, $H = \{h_1, h_2, \dots, h_m\}$ is a set consisting of haplotype segments of length L, and d is a nonnegative integer. The segments in D are from diseased individuals while those in N are from normal individuals. For an example of (D, N, H, d), see Fig. 1.

Recall that a haplotype segment is a binary string, while a genotype segment is a string on alphabet $\{0, 1, 2\}$. A *haplotype pair* for a genotype segment g is a pair (h, h') of haplotype segments of the same length as g such that the following conditions hold for every position q:

1. If g has a 0 or 1 at position q, then both h and h' have the same letter as g does at position q.
2. If g has a 2 at position q, then one of h and h' has a 0 at position q while the other has a 1 at position q.

$$
\begin{array}{l}
D \begin{cases}
g_1 : 22222222222222222222011000101000110 \\
g_2 : 22222222222222201001222220101000110 \\
g_3 : 22222222222222201001011002222200110 \\
g_4 : 22222222222222201001011000101022222
\end{cases} \\
N \begin{cases}
g_5 : 21022020212212021201011002222222021 \\
g_6 : 10220010101202101001222222222200110 \\
g_7 : 21022222021221212222222220101000110
\end{cases} \\
H \begin{cases}
h_1 : 11011010010011001001011000101000110 \\
h_2 : 00100101101100110110011000101000110 \\
h_3 : 10010110110010010110011000101000110 \\
h_4 : 11011010111010001001100110101000110 \\
h_5 : 01101011101001101001100110101000110 \\
h_6 : 01010111101100101001011001010100110 \\
h_7 : 01011110110010001001011001010100110 \\
h_8 : 11110101100111101001011000101011001 \\
h_9 : 11010110011111001001011000101011001
\end{cases} \\
d = 4
\end{array}
$$

Fig. 1 An example instance of the SC problem

$$\begin{aligned}
s &= 100110000100000\underline{01001011000101000110}\\
p &= 1\\
h_{1,2} &= 01100111101111110110011000101000110\\
h_{2,2} &= 01100111101111101001100110101000110\\
h_{3,2} &= 01100111101111101001011001010100110\\
h_{4,2} &= 01100111101111101001011000101011001\\
h_{5,1} &= 0100101011001000100101100\underline{1}010100011\\
h_{5,2} &= 110100000111110\underline{1}1101011000101011001\\
h_{6,1} &= 101000101010011010011\underline{0}0110101000110\\
h_{6,2} &= 1001001010110010100101100\underline{1}010100110\\
h_{7,1} &= 010001010011011\underline{1}0110011000101000110\\
h_{7,2} &= 110110100110110\underline{1}1001100110101000110
\end{aligned}$$

Fig. 2 A solution to the instance in Fig. 1, where the underlined part of s consists of those positions at which the letter of some $g_i \in D$ is 0 or 1, and the underlined letter of each $h_{i,j}$ with $5 \le i \le 7$ and $1 \le j \le 2$ differs from the letter of s at the same position

For example,

$$\begin{aligned}
&(10011000010000001001011000101000110,\\
&\ 01100111101111110110011000101000110)
\end{aligned}$$

is a haplotype pair for g_1 in Fig. 1.

Given an input (D, N, H, d), the SC problem requires the computation of a *solution* to (D, N, H, d) which consists of a *center haplotype segment* s, a *center index* $p \in \{1, 2, \dots, m\}$, and a haplotype pair $(h_{i,1}, h_{i,2})$ for each $g_i \in D \cup N$ such that the following conditions hold:

C1. $dist(s, h_p) \le d$.
C2. For each $i \in \{1, 2, \dots, k\}$, $h_{i,1} = s$ and there is an integer $x_i \in \{1, 2, \dots, m\}$ with $dist(h_{i,2}, h_{x_i}) \le d$.
C3. For each $i \in \{k+1, k+2, \dots, n\}$ and for each $j \in \{1, 2\}$, the following hold:

- C3a. There is an integer $x_{i,j} \in \{1, 2, \dots, m\} \setminus \{p\}$ with $dist(h_{i,j}, h_{x_{i,j}}) \le d$.
- C3b. There is at least one position q at which the letters of $h_{i,j}$ and s differ and the letter of some $g_\ell \in D$ is 0 or 1.

Note that the position q in Condition C3b depends not only on i and j but also on $h_{i,j}$, i.e., different i, j, or $h_{i,j}$ may yield different q. Figure 2 shows a solution to the example instance in Fig. 1, where the integers $x_1, \dots, x_4, x_{5,1}, x_{5,2}, \dots, x_{7,1}, x_{7,2}$ guaranteed in Conditions C2 and C3a are 2, 5, 6, 8, 7, 9, 5, 6, 2, 4, respectively.

4 The Hardness of the SC problem

A *parameterized problem* Q over an alphabet Σ is a subset of $\Sigma^* \times \mathbf{N}$, where Σ^* is the set of all strings over Σ and $\mathbf{N}$ is the set of all nonnegative integers. A parameterized problem Q over an alphabet Σ is *fixed-parameter tractable* if for every $(x,k) \in \Sigma^* \times \mathbf{N}$, we can decide whether $(x,k) \in Q$ or not in time $O(f(k) \cdot |x|^c)$ for some constant c and computable function f.

Let FPT denote the set of all fixed-parameter tractable problems. There are a number of problems that do not seem to belong to FPT. So, certain complexity classes have been defined to include such problems in the literature. $W[1]$ is one of them. Here, we omit the somewhat technical definition of $W[1]$. For a precise definition of $W[1]$, the reader is referred to [5, 6].

To give strong evidence that certain problems in $W[1]$ are unlikely to belong to FPT, the theory of $W[1]$-hardness has been developed. At the heart of this theory is the notion of parameterized reduction. A *parameterized reduction* from a parameterized problem Q over an alphabet Σ to another parameterized problem Q' over an alphabet Γ is a function that maps each pair $(x,k) \in \Sigma^* \times \mathbf{N}$ to a pair $(x',k') \in \Gamma^* \times \mathbf{N}$ such that the following conditions hold:

- $(x,k) \in Q$ if and only if $(x',k') \in Q'$.
- k' is bounded from above by a function of k.
- (x',k') can be computed in time $O(f(k) \cdot |x|^c)$ for some constant c and function f.

A parameterized problem Q' is $W[1]$-*hard* if for every parameterized problem Q in $W[1]$, there is a parameterized reduction from Q to Q'. A lot of parameterized problems have been proved to be $W[1]$-hard in the literature. In general, to prove a new parameterized problem Q to be $W[1]$-hard, it is known that we can proceed as follows. First, select a parameterized problem Q' that is known to be $W[1]$-hard. Then, give a parameterized reduction from Q' to Q.

We next show that the SC problem is $W[1]$-hard with parameters d and n.

Theorem 4.1 *The SC problem is $W[1]$-hard with parameters d and n.*

Proof We give a parameterized reduction from the binary closest-substring (BCSS) problem to the special case of the SC problem where all the individuals are diseased. Recall that an instance of the BCSS problem is a tuple $(s_1, \ldots, s_k, L, d)$, where $s_1, \ldots, s_k$ are binary strings each of length at least L and d is a nonnegative integer. Given $(s_1, \ldots, s_k, L, d)$, the BCSS problem asks if there is a binary string t of length L such that for all $1 \le i \le k$, s_i has a substring s'_i of length L with $dist(t, s'_i) \le d$. It is known that the BCSS problem is $W[1]$-hard with parameters d and k [14].

Let $(s_1, \ldots, s_k, L, d)$ be an instance of the BCSS problem. For each $1 \le i \le k$, let L_i be the length of s_i. For convenience, for a letter $\ell \in \{0,1,2\}$ and a nonnegative integer i, let ℓ^i denote the string consisting of i ℓs. Note that ℓ^0 is the empty string. We obtain $m = (L_1 - L + 1) + \sum_{i=1}^{k}(L_i - L + 1)$ strings $h_1, h_2, \ldots, h_m$ as follows:

1. For each $1 \le j \le L_1 - L + 1$, $h_j = s_1[j..j+L-1]0^{(d+1)k}$.
2. For each $i \in \{1, \ldots, k\}$ and each $1 \le j \le L_i - L + 1$, $h_y = \overline{s_i[j..j+L-1]} \times 0^{(d+1)(i-1)}1^{d+1}0^{(d+1)(k-i)}$, where $y = (L_1 - L + 1) + \sum_{z=1}^{i-1}(L_z - L + 1) + j$.

We further obtain k strings $g_1, \dots, g_k$ as follows:

- For each $i \in \{1, \dots, k\}$, $g_i = 2^L 0^{(d+1)(i-1)} 2^{d+1} 0^{(d+1)(k-i)}$.

Suppose that $(s_1, \dots, s_k, L, d)$ has a solution t in the BCSS problem. Then, for each $1 \le i \le k$, there is an integer j_i with $1 \le j_i \le L_i - L + 1$ such that $dist(t, s_i[j_i..j_i + L - 1]) \le d$. We next construct a solution for the instance $(\{g_1, \dots, g_k\}, \emptyset, \{h_1, \dots, h_m\}, d)$ of the SC problem as follows.

1. $s = t0^{(d+1)k}$. Note that $dist(s, h_{j_1}) \le d$ because $dist(t, s_1[j_1..j_1 + L - 1]) \le d$.
2. For each $i \in \{1, \dots, k\}$, construct a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i by setting $h_{i,1} = s$ and $h_{i,2} = \overline{t}0^{(d+1)(i-1)}1^{d+1}0^{(d+1)(k-i)}$. Note that for each $1 \le i \le k$, $dist(h_{i,2}, h_y) = dist(\overline{t}, \overline{s_i[j_i..j_i + L - 1]}) = dist(t, s_i[j_i..j_i + L - 1]) \le d$, where $y = (L_1 - L + 1) + \sum_{z=1}^{i-1}(L_z - L + 1) + j$.

Conversely, suppose that the instance $(\{g_1, \dots, g_k\}, \emptyset, \{h_1, \dots, h_m\}, d)$ of the SC problem has a solution. Let s be the center haplotype segment in the solution. Let t be the prefix of s with $|t| = L$. We claim that t is a solution to $(s_1, \dots, s_k, L, d)$ in the BCSS problem. To see this, first note that for each $1 \le i \le (d + 1)k$, there is an integer $j \in \{1, \dots, k\}$ such that the ith rightmost letter of g_j is a 0. This implies that the last $(d + 1)k$ bits of s are 0s. So, the string h_j with $dist(s, h_j) \le d$ has to be among $h_1, \dots, h_{L_1-L+1}$ because there are $d + 1$ 1s in the last $(d + 1)k$ bits of each h_j with $L_1 - L + 2 \le j \le m$. Thus, $dist(t, s_1[j..j + L - 1]) \le d$ for some $1 \le j \le L_1 - L + 1$. Moreover, for each $1 \le i \le k$, if we decompose g_i into two strings $h_{i,1}$ and $h_{i,2}$ with $h_{i,1} = s$, then $h_{i,2} = \overline{t}0^{(d+1)(i-1)}1^{d+1}0^{(d+1)(k-i)}$. Hence, for each $1 \le i \le k$, the string h_y with $1 \le y \le m$ and $dist(h_y, h_{i,2}) \le d$ has to satisfy $(L_1 - L + 1) + \sum_{z=1}^{i-1}(L_z - L + 1) + 1 \le y \le (L_1 - L + 1) + \sum_{z=1}^{i-1}(L_z - L + 1) + (L_i - L + 1)$, because of the different locations of the $d + 1$ 1s in the last $(d + 1)k$ bits of $h_1, \dots, h_m$. Therefore, for some $1 \le j \le L_i - L + 1$, $dist(t, s_i[j..j + L - 1]) = dist(\overline{t}, \overline{s_i[j..j - L + 1]}) \le d$. This completes the proof of the claim and hence that of the theorem. □

Corollary 4.2 *As long as the ETH conjecture is true, the SC problem cannot be solved in time $O(f(d, k)\tilde{n}^{o(\log d)})$ for any computable function f, where $\tilde{n}$ is the length of the input to the SC problem.*

Proof Marx [14] shows that as long as the ETH conjecture is true, the BCSS problem cannot be solved in $O(f(d, k)\hat{n}^{o(\log d)})$ time for any computable function f, where $\hat{n}$ is the length of the input to the BCSS problem. In the reduction described in the proof of Theorem 4.1, we constructed an instance I of the SC problem from a length-$\hat{n}$ instance of the BCSS problem such that $|I| = O(\hat{n}^2)$. Moreover, the parameters in the two instances are the same. Thus, the corollary holds. □

5 Exact Algorithms for the SC Problem

Throughout this section, let $\mathcal{I} = (D, N, H, d)$ be an instance of the SC problem, where $D = \{g_1, g_2, \dots, g_k\}$, $N = \{g_{k+1}, g_{k+2}, \dots, g_n\}$, and $H = \{h_1, h_2, \dots, h_m\}$.

Consider a genotype segment $g_i \in D \cup N$ and a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i. A position of g_i with a letter 0 indicates that both $h_{i,1}$ and $h_{i,2}$ have a letter 0 at the position, while a position of g_i with a letter 1 indicates that both $h_{i,1}$ and $h_{i,2}$ have a letter 1 at the position. On the other hand, a position of g_i with a letter 2 indicates that one of $h_{i,1}$ and $h_{i,2}$ has a 0 at the position while the other has a 1 at the position. For convenience, we say that a position of g_i is *decided* if the letter of g_i at the position is 0 or 1, and is *undecided* otherwise.

For D, we define three sets as follows:

- The set of *decided positions associated with D* consists of all positions q in R such that q is a decided position of at least one string in D.
- The set of *undecided positions associated with D* consists of all positions q in R such that q is an undecided position for all strings in D.
- The set of *conflicting positions associated with D* consists of all positions q in R such that q is a decided position of two distinct $g_i \in D$ and $g_j \in D$ but the letters of g_i and g_j at position q differ.

We say that an integer b is a *valid radius* if the instance (D, N, H, b) to the SC problem has a solution. Our goal is to decide if d is a valid radius. Obviously, the following condition is necessary for d to be a valid radius:

A1. The set of conflicting positions associated with D is empty.

So, we hereafter assume that Condition A1 holds.

For convenience, we define the following notations:

- L is the common length of the strings in $D \cup N \cup H$.
- U (respectively, $\overline{U}$) is the set of undecided (respectively, decided) positions associated with D.
- For each $g_i \in N$, U_i (respectively, $\overline{U_i}$) is the set of undecided (respectively, decided) positions of g_i.

Now, Condition C3b in Sect. 1 can be concisely rewritten as follows:

C3b. $h_{i,j}|_{\overline{U}} \neq s|_{\overline{U}}$.

Since Condition A1 holds, we can define a letter ℓ_q for each position $q \in \overline{U}$ as follows:

- If some segment in D is 0 at position q, then each of the other segments in D is 0 or 2 at position q and so we define $\ell_q = 0$.
- If some segment in D is 1 at position q, then each of the other segments in D is 1 or 2 at position q and so we define $\ell_q = 1$.

We call ℓ_q the *center letter* at position q.

Consider a $g_i \in N$. We say that g_i is *free* if there is a position in $\overline{U_i} \cap \overline{U}$ at which the center letter is different from the letter of g_i. On the other hand, we say that g_i is *dead* if (1) $|\overline{U} \setminus \overline{U_i}| \leq 1$ and (2) at every position q in $\overline{U_i} \cap \overline{U}$, the center letter is the same as the letter of g_i. For example, both g_5 and g_7 in Fig. 1 are free while g_6 is neither free nor dead.

In [12], it is shown that L is a valid radius only if the following condition holds:

A2. No string $g_i \in N$ is dead.

Since d is a valid radius only when L is a valid radius, we hereafter assume that Condition A2 holds.

5.1 Decomposing $g_i \in N$

Throughout this subsection, fix a genotype segment $g_i \in N$ and two haplotype segments h_{j_1} and h_{j_2} in H. Note that it is possible that $j_1 = j_2$. Our goal is to decide if there is a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying the following conditions:

B1. $dist(h_{i,1}, h_{j_1}) \le d$.
B2. $dist(h_{i,2}, h_{j_2}) \le d$.
B3. $h_{i,1}|_{\overline{U}} \ne s|_{\overline{U}}$.
B4. $h_{i,2}|_{\overline{U}} \ne s|_{\overline{U}}$.

To reach the above goal, we first define several notations:

- $d_1 = dist(g_i|_{\overline{U}_i}, h_{j_1}|_{\overline{U}_i})$.
- $d_2 = dist(g_i|_{\overline{U}_i}, h_{j_2}|_{\overline{U}_i})$.
- S is the set of positions $q \in U_i$ such that the letters of h_{j_1} and h_{j_2} at position q coincide.

Example 5.1 If $g_i = g_5$, $h_{j_1} = h_7$, and $h_{j_2} = h_9$ in Fig. 1, then $d_1 = 3$, $d_2 = 1$, and $S = \{4, 7, 16, 18\}$.

Lemma 5.1 *If at least one of the following conditions holds, then it is easy to decide if there is a haplotype pair* $(h_{i,1}, h_{i,2})$ *for* g_i *satisfying Conditions* B1 *through* B4.

1. $d_1 = d$ *or* $d_2 = d$.
2. $d_1 + d_2 + |S| > 2d$.
3. $d_1 < d$, $d_2 < d$, $d_1 + d_2 + |S| \le 2d$, *and* g_i *is free or* $(\overline{U} \cap U_i) \setminus S$ *contains two positions* q_1 *and* q_2 *such that the letter of* h_{j_1} *at position* q_1 *is not the center letter at position* q_1 *and the letter of* h_{j_2} *at position* q_2 *is not the center letter at position* q_2.

Proof Suppose that $d_1 = d$. Then, every haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Condition B1 must satisfy $h_{i,1} = h_{j_1}$. Note that there is a unique haplotype pair $(h_{i,1}, h_{i,2})$ for g_i with $h_{i,1} = h_{j_1}$. So, it suffices to check if this pair $(h_{i,1}, h_{i,2})$ satisfies Conditions B1 through B4. Obviously, this checking can be done in $O(L)$ time. Similar arguments apply when $d_2 = d$.

Next suppose that $d_1 + d_2 + |S| > 2d$. We claim that there is no haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Conditions B1 and B2. To this claim, first observe that for each position $q \in S$, either the letters of $h_{i,1}$ and h_{j_1} at position q differ or the letters of $h_{i,2}$ and h_{j_2} at position q differ. Thus, the positions in S contribute $|S|$ to $dist(h_{i,1}, h_{j_1}) + dist(h_{i,2}, h_{j_2})$. We also know that the positions outside S contribute at least $d_1 + d_2$ to $dist(h_{i,1}, h_{j_1}) + dist(h_{i,2}, h_{j_2})$. Hence,

$dist(h_{i,1}, h_{j_1}) + dist(h_{i,2}, h_{j_2}) > 2d$. Consequently, it is impossible for $(h_{i,1}, h_{i,2})$ to satisfy Conditions B1 and B2.

Finally, suppose that $d_1 < d$, $d_2 < d$, $d_1 + d_2 + |S| \le 2d$, and g_i is free or $(\overline{U} \cap U_i) \setminus S$ contains two distinct positions q_1 and q_2 such that the letter of h_{j_1} at position q_1 is not the center letter at position q_1 and the letter of h_{j_2} at position q_2 is not the center letter at position q_2. Then, we can arbitrarily partition S into two subsets S_1 and S_2 such that $d_1 + |S_1| \le d$ and $d_2 + |S_2| \le d$. For example, we can let $S_1 = \{4\}$ and $S_2 = \{7, 16, 18\}$ in Example 5.1 because g_5 is free. We can now construct a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i as follows. For each position $q \in \overline{U_i}$, let the letters of $h_{i,1}$ and $h_{i,2}$ at position q be the letter of g_i at position q. For each position $q \in U_i \setminus S$, let the letter of $h_{i,1}$ (respectively, $h_{i,2}$) at position q be the letter of h_{j_1} (respectively, h_{j_2}) at position q. For each position $q \in S_1$ (respectively, $q \in S_2$), let the letter of $h_{i,1}$ at position q be different from (respectively, the same as) the letter of h_{j_1} at position q, while let the letter of $h_{i,2}$ at position q be the same as (respectively, different from) the letter of h_{j_2} at position q. For example, if we partition S in Example 5.1 into $S_1 = \{4\}$ and $S_2 = \{7, 16, 18\}$, then $h_{5,1}$ and $h_{5,2}$ are as shown in Fig. 2. Obviously, $(h_{i,1}, h_{i,2})$ satisfies Conditions B1 through B4 and can be constructed in $O(L)$ time. □

By Lemma 5.1 and Condition A2, we may assume that the following hold:

D1. $d_1 < d$ and $d_2 < d$.
D2. $d_1 + d_2 + |S| \le 2d$.
D3. g_i is neither free nor dead.
D4. $(\overline{U} \cap U_i) \setminus S$ does not contain two positions q_1 and q_2 such that the letter of h_{j_1} at position q_1 is not the center letter at position q_1 and the letter of h_{j_2} at position q_2 is not the center letter at position q_2.

By Condition D3, $|\overline{U} \cap U_i| \ge 2$. Without loss of generality, we assume that the center letters at the positions in $\overline{U} \cap U_i$ are all 0s.

Lemma 5.2 *If at least one of the following two conditions holds, then there is no haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Conditions* B1 *through* B4.

E1. $|\overline{U} \cap U_i| = 2$, $\overline{U} \cap U_i \subseteq S$, *the two letters of h_{j_1} at positions in $\overline{U} \cap U_i$ are different, and $d_1 = d_2 = d - 1$.*
E2. $d_1 + d_2 + |S| \ge 2d - 1$, $\overline{U} \cap S = \emptyset$, *and either the letters of h_{j_1} at the positions in $\overline{U} \cap U_i$ are all* 0*s or the letters of h_{j_2} at the positions in $\overline{U} \cap U_i$ are all* 0*s.*

Otherwise, we can find a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Conditions B1 *through* B4 *in $O(L)$ time.*

Proof Suppose that Condition E1 holds. Towards a contradiction, assume that there is a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Conditions B1 through B4. Then, since the two center letters at the positions in $\overline{U} \setminus \overline{U_i}$ are both 0s, Conditions B3, B4, and D3 together imply that either $h_{i,1}|_{\overline{U} \setminus \overline{U_i}} = 01$ and $h_{i,2}|_{\overline{U} \setminus \overline{U_i}} = 10$, or $h_{i,1}|_{\overline{U} \setminus \overline{U_i}} = 10$ and $h_{i,2}|_{\overline{U} \setminus \overline{U_i}} = 01$. We assume that $h_{i,1}|_{\overline{U} \setminus \overline{U_i}} = 01$ and $h_{i,2}|_{\overline{U} \setminus \overline{U_i}} = 10$; the other case is similar. On the other hand, since the two letters of h_{j_1} at positions in $\overline{U} \cap U_i$ are different, either $h_{j_1}|_{\overline{U} \setminus \overline{U_i}} = h_{j_2}|_{\overline{U} \setminus \overline{U_i}} = 01$ or $h_{j_1}|_{\overline{U} \setminus \overline{U_i}} = h_{j_2}|_{\overline{U} \setminus \overline{U_i}} = 10$. In the former case, the two letters of $h_{i,2}$ at positions in $\overline{U} \setminus \overline{U_i}$ are both different from the letters of h_{j_2} at the same positions. Moreover, in the latter case, the two letters of $h_{i,1}$ at positions in $\overline{U} \setminus \overline{U_i}$ are both different from the letters of h_{j_1} at the same positions. So, in both cases, $dist(h_{i,1}, h_{j_1}) \geq d_1 + 2 > d$ or $dist(h_{i,1}, h_{j_2}) \geq d_2 + 2 > d$, a contradiction against Conditions B1 and B2.

Suppose that Condition E2 holds. Consider an arbitrary haplotype pair $(h_{i,1}, h_{i,2})$ for g_i. As observed in the second paragraph of the proof of Lemma 5.1, the positions outside $U_i \cap \overline{U}$ already contribute at least $2d - 1$ to $dist(h_{i,1}, h_{j_1}) + dist(h_{i,2}, h_{j_2})$ because $\overline{U} \cap S = \emptyset$. So, $dist(h_{j_1}|_{U_i \cap \overline{U}}, h_{i,1}|_{U_i \cap \overline{U}}) = 0$ or $dist(h_{j_2}|_{U_i \cap \overline{U}}, h_{i,2}|_{U_i \cap \overline{U}}) = 0$; otherwise, $(h_{i,1}, h_{i,2})$ does not satisfy Condition B1 or B2. But then, Condition E2 implies that either the letters of $h_{i,1}$ at the positions in $\overline{U} \cap U_i$ are all 0s or the letters of $h_{i,2}$ at the positions in $\overline{U} \cap U_i$ are all 0s. In either case, $(h_{i,1}, h_{i,2})$ does not satisfy Condition B3 or B4.

Next, suppose that neither Condition E1 nor Condition E2 holds. We want to construct a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Conditions B1 through B4. To this end, we refine the construction in the last paragraph of the proof of Lemma 5.1, by distinguish several cases as follows.

Case 1: $d_1 + d_2 + |S| \geq 2d - 1$. In this case, since Condition E2 does not hold, Condition D4 implies that $\overline{U} \cap S \neq \emptyset$. We refine the construction in the last paragraph of the proof of Lemma 5.1, by distinguish five cases as follows.

Case 1.1: $|\overline{U} \cap S| = 1$. Then, since $|U_i \cap \overline{U}| \geq 2$, $(U_i \cap \overline{U}) \setminus S$ contains at least one position q. Obviously, either the letter of h_{j_1} at position q is a 1 or the letter of h_{j_2} at position q is a 1. We assume that the letter of h_{j_1} at position q is a 1; the other case is similar. Then, when partitioning S into S_1 and S_2, we put the following restriction: If the letter of h_{j_1} at the unique position $q' \in \overline{U} \cap S$ is a 0, $q' \in S_2$; otherwise, $q' \in S_1$.

Case 1.2: $\overline{U} \cap S$ contains two positions q_1 and q_2 such that the two letters of h_{j_1} at positions q_1 and q_2 are identical. In this case, when partitioning S into S_1 and S_2, we also require that $|\{q_1, q_2\} \cap S_1| = 1$.

Case 1.3: $|\overline{U} \cap S| \geq 2$, $d_1 < d - 1$, and the letters of h_{j_1} at the positions in $\overline{U} \cap S$ are distinct. In this case, $|\overline{U} \cap S| = 2$. Let q_1 and q_2 be the positions in $\overline{U} \cap S$. Then, when partitioning S into S_1 and S_2, we also require that $\{q_1, q_2\} \subseteq S_1$.

Case 1.4: $|\overline{U} \cap S| \geq 2$, $d_2 < d - 1$ and the letters of h_{j_1} at the positions in $\overline{U} \cap S$ are distinct. This case is the same as Case 1.3 except that we require $\{q_1, q_2\} \subseteq S_2$.

Case 1.5: None of Cases 1.1 through 1.4 occurs. Then, $|\overline{U} \cap S| = 2$, $d_1 = d_2 = d - 1$, and the two letters of h_{j_1} at the positions in $\overline{U} \cap S$ are different. In turn, since Condition E1 does not hold, $(\overline{U} \cap U_i) \setminus S \neq \emptyset$. Let q_0 (respectively, q_1) be the position in $\overline{U} \cap S$ at which the letter of h_{j_1} is a 0 (respectively, 1). Let q be an arbitrary position in $(\overline{U} \cap U_i) \setminus S$. Then, when partitioning S into S_1 and S_2, we put the following restriction: If the letter of h_{j_1} at position q is a 1, then $q_1 \in S_1$ and $q_0 \in S_2$; otherwise, $q_1 \in S_2$ and $q_0 \in S_1$.

Case 2: $d_1 + d_2 + |S| \leq 2d - 2$. In this case, we refine the construction in the last paragraph of the proof of Lemma 5.1, by distinguish four cases as follows.

Case 2.1: $|(\overline{U} \cap U_i) \setminus S| \geq 2$. In this case, we select an arbitrary position q in $(\overline{U} \cap U_i) \setminus S$. By Condition D4, either the letters of h_{j_1} at the positions in $(\overline{U} \cap U_i) \setminus S$ are all 0s or the letters of h_{j_2} at the positions in $(\overline{U} \cap U_i) \setminus S$ are all 0s. When partitioning S into S_1 and S_2, we also require that $d_1 + |S_1| \leq d - 1$ and $d_2 + |S_2| \leq d - 1$. Moreover, when constructing the haplotype pair $(h_{i,1}, h_{i,2})$ for g_i, we fix the letters of $h_{i,1}$ and $h_{i,2}$ at the positions in $U_i \setminus S$ as follows. As before, for each position $q' \in U_i \setminus (S \cup \{q\})$, let the letter of $h_{i,1}$ (respectively, $h_{i,2}$) at position q' be the letter of h_{j_1} (respectively, h_{j_2}) at position q'. However, let the letter of $h_{i,1}$ (respectively, $h_{i,2}$) at position q be different from the letter of h_{j_1} (respectively, h_{j_2}) at position q. This change yields that $dist(h_{i,1}, h_{j_1}) = d_1 + |S_1| + 1 \leq d$ and $dist(h_{i,2}, h_{j_2}) = d_2 + |S_2| + 1 \leq d$.

Case 2.2: $|(\overline{U} \cap U_i) \setminus S| = 1$. In this case, let q_1 be the unique position in $(\overline{U} \cap U_i) \setminus S$. Note that either the letter of h_{j_1} at position q_1 is a 1 or the letter of h_{j_2} at position q_1 is a 1. We assume that the letter of h_{j_1} at position q_1 is a 1; the other case is similar. Let q_2 be an arbitrary position in $\overline{U} \cap S$. Then, when partitioning S into S_1 and S_2, we put the following restriction: If the letter of h_{j_2} at position q_2 is a 0, then $q_2 \in S_2$; otherwise, $q_2 \in S_1$.

Case 2.3: $\overline{U} \cap S$ contains two positions q_1 and q_2 such that the two letters of h_{j_1} at positions q_1 and q_2 are identical. This case is the same as Case 1.2.

Case 2.4: None of Cases 2.1 through 2.3 occurs. In this case, $\overline{U} \cap U_i \subseteq S$, $|\overline{U} \cap U_i| = 2$, and the letters of h_{j_1} at the two positions in $\overline{U} \cap U_i$ are different. In turn, since Condition E1 does not hold, $d_1 \leq d - 2$ or $d_2 \leq d - 2$. We assume that $d_1 \leq d - 2$; the other case is similar. Let q_1 and q_2 be the two positions in $\overline{U} \cap U_i$. Then, when partitioning S into S_1 and S_2, we also require that $\{q_1, q_2\} \subseteq S_1$. $\square$

For convenience, we say that an integer $p \in \{1, \dots, m\}$ is *valid* for a $g_i \in N$ if there is a haplotype pair $(h_{i,1}, h_{i,2})$ for g_i satisfying Condition C3. Now, we are ready to state the key lemma in this subsection.

Lemma 5.3 *Given an integer* $p \in \{1, \dots, m\}$, *we can decide in* $O(m^2 L(n-k))$ *time if* p *is valid for every* $g_i \in N$.

5.2 Decomposing the Strings in D

Throughout this subsection, fix an integer $p \in \{1, 2, \dots, m\}$ that is valid for every $g_i \in N$. Let s_p be the haplotype segment constructed by letting $s_p[q]$ be the center letter at position q for each $q \in \overline{U}$, and letting $s_p[q] = h_p[q]$ for each $q \in U$. Obviously, if t is a binary string with $|t| = L$ and $t|_{\overline{U}} = s_p|_{\overline{U}}$, then for each $g_i \in D$, there is a unique haplotype pair $(h_{i,1}, h_{i,2})$ with $h_{i,1} = t$. So, we use $\overline{t(i)}$ to denote this $h_{i,2}$.

Let $d_p = dist(s_p|_{\overline{U}}, h_p|_{\overline{U}})$. Our task is to decide if we can obtain a string t by modifying at most $d - d_p$ letters of s_p at the positions in U so that for each $g_i \in D$, there is an integer $x_i \in \{1, 2, \dots, m\}$ with $dist(\overline{t(i)}, h_{x_i}) \leq d$. Our task becomes much easier if we know the integer x_i for each $g_i \in D$. So, we consider this case first.

```
Input:  A triple $(s, P, b)$, where $s$ is a string of length $L$ with $s|_{\overline{U}} = s_p|_{\overline{U}}$,
        $P$ is a subset of $[1..L]$ with $\overline{U} \subseteq P$, and $b$ is a nonnegative integer less
        than or equal to $d - d_p$.
Output: A string $t$ obtained by modifying at most $b$ positions of $s$ in $U \setminus P$
        such that $dist(\overline{t(i)}, h_{x_i}) \le d$ for each $g_i \in D$, if such a $t$ exists;
        nothing, otherwise.
1. If $dist(h_{x_i}, \overline{s(i)}) \le d$ for every $g_i \in D$, then output $s$ and stop immediately.
2. Select an arbitrary $g_i \in D$ with $dist(h_{x_i}, \overline{s(i)}) > d$.
3. If $dist(\overline{s(i)}, h_{x_i}) - d > \min\{b, |\{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P|\}$, then return.
4. Let $Q = \{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P$ and $\ell = dist(\overline{s(i)}, h_{x_i}) - d$.
5. Guess a subset $Z$ of $Q$ with $\ell \le |Z| \le b$.
6. Obtain a string $s'$ by modifying $s$ by flipping the letters at the positions in $Z$.
7. Recursively call the algorithm on input $(s', P \cup Q, \min\{b - |Z|, |Z| - \ell\})$.
```

Fig. 3 Algorithm 1 for the SC problem

5.2.1 The Case When $x_1, \ldots, x_k$ in Condition C2 *Are Known*

In this case, we want to decide if we can obtain a string t by modifying at most $d - d_p$ letters of s_p at the positions in U so that $dist(\overline{t(i)}, h_{x_i}) \le d$. This case resembles the *binary closest string* (BCS) problem. Recall that an instance of the BCS problem is a pair $(\mathcal{S}, d)$, where $\mathcal{S}$ is a set of binary strings of the same length L and d is a nonnegative integer. Given $(\mathcal{S}, d)$, the BCS problem asks if there is a binary string t of length L such that $dist(t, s_i) \le d$ for all $s_i \in \mathcal{S}$. Known algorithms for the BCS problem can be found in [2, 3, 7, 13, 17, 20]. All the algorithms indeed solve a more general problem (called the *extended BCS* problem). An input to the extended BCS problem contains not only $(\mathcal{S}, d)$ but also a triple (s, P, b), where s is a string of length L, P is a subset of $[1..L]$, and b is an integer less than or equal to d. The objective is to decide if we can modify at most b letters of s at the positions in $[1..L] \setminus P$ so that $dist(s, s_i) \le d$ for all $s_i \in \mathcal{S}$.

The correspondence between the extended BCS problem and the special case of the SC problem is as follows: s, b, $s_i \in \mathcal{S}$, and P in the former correspond to s_p, $d - d_p$, h_{x_i}, and $\overline{U}$ in the latter, respectively. A slight difference between the two is that the former tests if $dist(s, s_i) \le d$ for all $s_i \in \mathcal{S}$, while the latter tests if $dist(\overline{s_p(i)}, h_{x_i}) \le d$ for all $g_i \in D$. Based on this correspondence and difference, it is easy to modify the algorithm in [13] for the extended BCS problem so that it works for the special case of the SC problem. The resulting algorithm (called *Algorithm 1*) is shown in Fig. 3.

To solve our special case, it suffices to call Algorithm 1 on input $(s_p, \overline{U}, d - d_p)$. The correctness of Algorithm 1 relies on the following lemma:

Lemma 5.4 *Let (s, P, b) be an input to Algorithm* 1. *Assume that t is an output of Algorithm* 1 *on input (s, P, b). Suppose that $dist(\overline{s(i)}, h_{x_i}) > d$ for some $g_i \in D$. Let $\ell = dist(\overline{s(i)}, h_{x_i}) - d$, z be the number of positions $q \in \{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P$ with $\overline{t(i)}[q] \ne \overline{s(i)}[q]$, and b' be the number of positions $q \in [1..L] \setminus (P \cup \{\overline{s(i)} \not\equiv h_{x_i}\})$ with $\overline{t(i)}[q] \ne \overline{s(i)}[q]$. Then, $b' \le \min\{b - z, z - \ell\}$. Consequently, $b' \le \frac{1}{2}(b - \ell)$.*

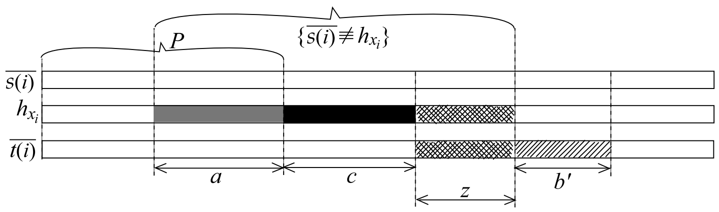

Fig. 4 Strings $\overline{s(i)}$, h_{x_i}, and $\overline{t(i)}$ in Lemma 5.4, where for each position $i \in [1..L]$, two of the strings have the same letter at position i if and only if they are illustrated in the same color or pattern at position i

Proof The proof is the same as that of Lemma 3.1 in [2]. To be self-contained, we include it here.

Obviously, $d(\overline{s(i)}, \overline{t(i)}) = z + b'$ (see Fig. 4). Since t is an output of Algorithm 1 on input (s, P, b), $dist(\overline{t(i)}, \overline{s(i)}) \le b$. Thus, $b' \le b - z$.

Let $a = |P \cap \{\overline{t(i)} \not\equiv h_{x_i}\}|$, and let c be the number of positions $q \in \{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P$ with $\overline{s(i)}[q] = \overline{t(i)}[q]$. Then, $|\{\overline{s(i)} \not\equiv h_{x_i}\}| = a + c + z$ and $dist(\overline{t(i)}, h_{x_i}) = a + c + b'$. So, by assumption, $a + c + z = d + \ell$ and $a + c + b' \le d$. Thus, $b' \le z - \ell$. □

To see the correctness of Algorithm 1, first observe that Step 1 is clearly correct. To see that Step 3 is also correct, first note that $dist(h_{x_i}, \overline{s(i)}) = |\{h_{x_i} \not\equiv \overline{s(i)}\}| = d + \ell$. So, in order to satisfy $dist(h_{x_i}, \overline{s(i)}) \le d$, we need to first select at least ℓ positions among the positions in $\{h_{x_i} \not\equiv \overline{s(i)}\}$ and then modify the letters at the selected positions. By definition, we are allowed to select at most b positions and the selected positions have to be in $Q = \{h_{x_i} \not\equiv \overline{s(i)}\} \setminus P$; so no solution exists if $\ell > \min\{b, |Q|\}$. The correctness of Step 7 is guaranteed by Lemma 5.4. This can be seen by viewing $|Z|$ in Algorithm 1 as z in Lemma 5.4, and viewing b' in Lemma 5.4 as the number of positions (outside $P \cup \{h_{x_i} \not\equiv \overline{s(i)}\} = P \cup Q$) of $\overline{s(i)}$ where the letters can be modified in order to transform $\overline{s(i)}$ into $\overline{t(i)}$. That is, $\min\{b - |Z|, |Z| - \ell\}$ in Step 7 corresponds exactly to $\min\{b - z, z - \ell\}$ in Lemma 5.4.

The execution of Algorithm 1 on input (s, P, b) can be modeled by a tree $\mathcal{T}$ in which the root corresponds to (s, P, b), each other node corresponds to a recursive call, and a recursive call A is a child of another call B if and only if B calls A directly. We call $\mathcal{T}$ the *search tree* on input (s, P, b). By the construction of Algorithm 1, each non-leaf node in $\mathcal{T}$ has at least two children. Thus, the number of nodes in $\mathcal{T}$ is at most twice the number of leaves in $\mathcal{T}$. Consequently, we can focus on how to bound the number of leaves in $\mathcal{T}$. For convenience, we define the *size* of $\mathcal{T}$ to be the number of its leaves.

Let $T_1(d, d - d_p)$ be the size of the search tree of Algorithm 1 on input $(s_p, \overline{U}, d - d_p)$. Similar to Theorem 3.4 in [2], we can prove the next lemma:

Lemma 5.5 $T_1(d, d - d_p) \le \frac{(6\sqrt{3})^d}{(\sqrt{3} \cdot \sqrt[3]{4})^{d_p}}$.

As observed in [17], Algorithm 1 can be made faster by replacing Step 2 with the step in Fig. 5.

We call the modified algorithm *Algorithm 2*. The intuition behind Algorithm 2 is as follows. By Lemma 5.4, the larger ℓ is, the smaller b' is. Note that b' means the

2′. If $b = d - d_p$, then select a $g_i \in D$ such that $dist(h_{x_i}, \overline{s(i)}) \geq dist(h_{x_j}, \overline{s(j)})$ for all $g_j \in D$. Otherwise, select an arbitrary $g_i \in D$ with $dist(h_{x_i}, \overline{s(i)}) > d$.

Fig. 5 Obtaining Algorithm 2 by modifying Step 2 of Algorithm 1

2″. If $b = d - d_p$, then *guess* a $g_i \in D$ and make a copy s_0 of s and a copy i_0 of i. Otherwise, select an arbitrary $g_i \in D$ with $dist(h_{x_i}, \overline{s(i)}) > d$ and $dist(h_{x_{i_0}}, \overline{s_0(i_0)}) \geq dist(h_{x_i}, \overline{s_0(i)})$.

Fig. 6 Obtaining Algorithm 3 by modifying Step 2′ of Algorithm 2

number of letters of $\overline{s(i)}$ we need to further modify. Thus, by maximizing ℓ, we can make the algorithm run faster.

Let $T_2(d, d - d_p)$ be the size of the search tree of Algorithm 2 on input $(s_p, \overline{U}, d - d_p)$. Similar to Theorem 4.3 in [2], we can prove the next lemma:

Lemma 5.6 $T_2(d, d - d_p) \leq \frac{8^d}{2^{d_p}}$.

We next obtain a slower version of Algorithm 2 by replacing Step 2′ with the step in Fig 6.

We call the modified algorithm *Algorithm 3*. Algorithm 3 will be useful later in this paper when we consider the general case where $x_1, \ldots, x_k$ are not known. Basically, if $b = d - d_p$, a string $g_i \in D$ in Step 2′ is hard to find when $x_1, \ldots, x_k$ are not known. In this case, our idea is to guess this $g_i \in D$ and use i_0 and s_0 to memorize i and s (for later use), respectively. Note that Algorithm 3 does not verify that for all $g_j \in D$, $dist(h_{x_{i_0}}, \overline{s_0(i_0)}) \geq dist(h_{x_j}, \overline{s_0(j)})$. Indeed, only for those g_i selected in Step 2″ of subsequent recursive calls, Algorithm 3 verifies that $dist(h_{x_{i_0}}, \overline{s_0(i_0)}) \geq dist(h_{x_i}, \overline{s_0(i)})$.

Algorithm 3 on input $(s_p, \overline{U}, d - d_p)$ is clearly correct, because (1) it performs the *guess* operation for $b = d - d_p$ by trying all $g_i \in D$ and (2) when trying the $g_i \in D$ with $dist(h_{x_i}, \overline{s_p(i)}) \geq dist(h_{x_j}, \overline{s_p(j)})$, it does the same as Algorithm 2. To estimate the running time of Algorithm 3 on input $(s_p, \overline{U}, d - d_p)$, let $T_3(d, b)$ be the size of the search tree of Algorithm 3 on input $(s_p, \overline{U}, d - d_p)$.

Lemma 5.7 $T_3(d, d - d_p) \leq k \cdot \frac{8^d}{2^{d_p}}$.

Proof Suppose that we modify Algorithm 3 on input $(s_p, \overline{U}, d - d_p)$ by not *guessing* $g_i \in D$ but rather choosing a particular $g_i \in D$. Let $\mathcal{T}_i$ be the search tree of the modified algorithm on input $(s_p, \overline{U}, d - d_p)$. Obviously, $T_3(d, d - d_p)$ does not exceed the total size of $\mathcal{T}_1, \ldots, \mathcal{T}_n$. So, it suffices to show that for each $i \in \{1, \ldots, k\}$, the size of $\mathcal{T}_i$ is at most $8^d/2^{d_p}$.

Fix an $i_0 \in \{1, \ldots, k\}$. To show that the size of $\mathcal{T}_{i_0}$ is at most $8^d/2^{d_p}$, first note that for each non-root node μ of $\mathcal{T}_{i_0}$, the recursive call (of the modified algorithm)

```
Input:   Same as that of Algorithm 1.
Output:  A string $t$ obtained by modifying at most $b$ positions of $s$ in $U \setminus P$
         such that for each $g_i \in D$, there is an integer $x_i \in \{1, \dots, m\}$ with
         $dist(\overline{t(i)}, h_{x_i}) \le d$, if such a $t$ exists; nothing, otherwise.
  1. If for every $g_i \in D$, there is a string $h_j \in H$ such that $dist(h_j, \overline{s(i)}) \le d$,
     then output $s$ and stop immediately.
  2. Select a $g_i \in D$ such that for every $h_j \in H$, $dist(h_j, \overline{s(i)}) > d$.
  3. If all $h_j \in H$ satisfy $dist(\overline{s(i)}, h_j) - d > \min\{b, |\{\overline{s(i)} \not\equiv h_j\} \setminus P|\}$, then
     return.
  4. Guess an $h_{x_i} \in H$ such that $dist(\overline{s(i)}, h_{x_i}) - d \le \min\{b, |\{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P|\}$.
5–7. Same as Steps 4, 5, and 6 in Algorithm 1, respectively.
  8. Recursively call the algorithm on input $(s', P \cup Q, \min\{b - |Z|, |Z| - \ell\})$.
```

Fig. 7 Algorithm 4 for the SC problem

corresponding to μ selects a $g_i \in D$ in Step $2''$ such that $dist(h_{x_i}, \overline{s(i)}) > d$ and $dist(h_{x_{i_0}}, \overline{s_0(i_0)}) \ge dist(h_{x_i}, \overline{s_0(i)})$. Because of these two inequalities, we can use similar arguments to those in the proofs of Lemmas 4.1 and 4.2 and Theorem 4.3 in [2] to prove that the size of $\mathcal{T}_{i_0}$ is at most $8^d/2^{d_p}$. □

5.2.2 The General Case

Here, we extend Algorithms 1 and 3 so that they work for the (general) SC problem. Motivated by an idea in [13], we extend Algorithm 1 by *guessing* x_i in Step 4. That is, we do not guess all of $x_1, \dots, x_k$ in advance. Rather, we guess x_i dynamically. The algorithm (called *Algorithm 4*) is shown in Fig. 7.

Obviously, if we do not guess h_{x_i} in Step 4 of Algorithm 4 but rather choose an arbitrary h_{x_i} in H there, then the search tree of Algorithm 4 on input (s, P, b) has the same size as the search tree of Algorithm 1 on input (s, P, b) does. So, to estimate the size of the search tree of Algorithm 4 on input (s, P, b), it suffices to find out how the guesses in Step 4 of Algorithm 4 expand the size of the search tree of Algorithm 1 on input (s, P, b). Clearly, the "guess" operation in Step 4 requires Algorithm 4 to try all $h_j \in H$ with $dist(\overline{s(i)}, h_j) - d \le \min\{b, |\{\overline{s(i)} \not\equiv h_j\} \setminus P|\}$. A single "guess" expands the size of the search tree by a factor of at most m. Because of Lemma 5.4, the recursion depth of Algorithm 4 is at most $\lfloor \log_2(b+1) \rfloor$. Thus, the size of the search tree of Algorithm 4 on input (s, P, b) is at most $m^{\lfloor \log_2(b+1) \rfloor}$ times that of the search tree of Algorithm 1 on input (s, P, b).

Now, let $T_4(d, d - d_p)$ be the size of the search tree of Algorithm 4 on input $(s_p, \overline{U}, d - d_p)$. Then,

Lemma 5.8 $T_4(d, d - d_p) \le \frac{(6\sqrt{3})^d}{(\sqrt{3} \cdot \sqrt[3]{4})^{d_p}} \cdot m^{\lfloor \log_2(d - d_p + 1) \rfloor}$.

Theorem 5.9 *Algorithm* 4 *takes* $O(kmL + kmd \cdot \frac{(6\sqrt{3})^d}{(\sqrt{3} \cdot \sqrt[3]{4})^{d_p}} \cdot m^{\lfloor \log_2(d - d_p + 1) \rfloor})$ *time.*

```
Input:   Same as that of Algorithm 1.
Output:  Same as that of Algorithm 4.
   1. If for every $g_i \in D$, there is a string $h_j \in H$ such that $dist(h_j, \overline{s(i)}) \leq d$,
      then output $s$ and stop immediately.
   2. If $b = d - d_p$, then guess a $g_i \in D$ and make a copy $s_0$ of $s$ and a copy $i_0$ of
      $i$. Otherwise, select a $g_i \in D$ such that for every $h_j \in H$, $dist(h_j, \overline{s(i)}) > d$.
   3. If all $h_j \in H$ satisfy $dist(\overline{s(i)}, h_j) - d > \min\{b, |\{\overline{s(i)} \not\equiv h_j\} \setminus P|\}$, then
      return.
   4. If $b = d - d_p$, guess an $h_{x_{i_0}} \in H$ such that $dist(\overline{s(i_0)}, h_{x_{i_0}}) - d \leq$
      $\min\{b, |\{\overline{s(i_0)} \not\equiv h_{x_{i_0}}\} \setminus P|\}$. Otherwise, guess an $h_{x_i} \in H$ such that
      $dist(\overline{s(i)}, h_{x_i}) - d \leq \min\{b, |\{\overline{s(i)} \not\equiv h_{x_i}\} \setminus P|\}$ and $dist(\overline{s_0(i_0)}, h_{x_{i_0}}) \geq$
      $dist(\overline{s_0(i)}, h_{x_i})$.
 5–7. Same as Steps 4, 5, and 6 of Algorithm 1, respectively.
   8. Recursively call the algorithm on input $(s', P \cup Q, \min\{b - |Z|, |Z| - \ell\})$.
```

Fig. 8 Algorithm 5 for the SC problem

Proof Obviously, excluding the recursive calls, each step of Algorithm 4 takes $O(kmL)$ time. To improve this time bound to $O(kmd)$, the idea is to perform an $O(kmL)$-time preprocessing. In the preprocessing, for each pair (i, j) with $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, m\}$, we compute $\Delta_{i,j} = \{\overline{s_p(i)} \not\equiv h_j\}$ and discard it if $|\Delta_{i,j}| > 2d - d_p$. Note that we modify only $O(d)$ letters of s_p at Step 7 of Algorithm 4 on input $(s_p, \overline{U}, d - d_p)$ and hence we can accordingly update each remaining $\Delta_{i,j}$ within $O(d)$ time. This is also true in subsequent recursive calls. So, by Lemma 5.8, the total time complexity of Algorithm 4 is as stated in the theorem. □

By Lemma 5.3 and Theorem 5.9, we have the following corollary immediately:

Corollary 5.10 *The SC problem can be solved in time*

$$O\left(m^3 L(n-k) + m^2 Lk + kd \cdot \frac{(6\sqrt{3})^d}{(\sqrt{3} \cdot \sqrt[3]{4})^{d'}} \cdot m^{\lfloor \log_2(d-d'+1) \rfloor + 2}\right),$$

where $d' = \min_{p \in \{1,\ldots,m\}} dist(s_p|_{\overline{U}}, h_p|_{\overline{U}})$.

We next extend Algorithm 3 so that it works for the general case. The idea is the same as that used to obtain Algorithm 4 from Algorithm 1. That is, as in Algorithm 4, we *guess* x_i dynamically in Step 4. The resulting algorithm (called *Algorithm 5*) is shown in Fig. 8.

Let $T_5(d, d - d_p)$ be the size of the search tree of Algorithm 5 on input $(s_p, \overline{U}, d - d_p)$. Then, as we obtained Lemma 5.8 from Lemma 5.5, we can obtain the next lemma from Lemma 5.7:

Lemma 5.11 $T_5(d, d - d_p) \leq k \cdot \frac{8^d}{2^{d_p}} \cdot m^{\lfloor \log_2(d - d_p + 1) \rfloor}$.

Using Lemma 5.11, we can prove the next theorem (whose proof is similar to that of Theorem 5.9):

Theorem 5.12 *Algorithm* 5 *takes* $O(kmL + k^2md \cdot \frac{8^d}{2^{d_p}} \cdot m^{\lfloor \log_2(d-d_p+1)\rfloor})$ *time.*

By Lemma 5.3 and Theorem 5.12, we have the following corollary immediately:

Corollary 5.13 *The SC problem can be solved in time*

$$O\left(m^3L(n-k) + m^2Lk + k^2d \cdot \frac{8^d}{2^{d'}} \cdot m^{\lfloor \log_2(d-d'+1)\rfloor+2}\right).$$

Roughly speaking, Algorithm 5 is faster than Algorithm 4 if and only if $d \geq \log_b k$, where $b = \frac{3\sqrt{3}}{4}$.

6 Experimental Results

We have implemented Algorithms 4 and 5 in C++ and used them to form a software package for linkage analysis for closely related individuals without a known pedigree.

Recall that to identify the true mutation regions of a chromosome C, we break C into a set R of length-500 segments [12]. For each segment $r \in R$, we can obtain an instance (D_r, N_r, H_r, d) of the SC problem and we first call the algorithm for the SC problem proposed in [12] (APPROX for short) on input $(D_r, N_r, H_r, 2d)$. If the algorithm outputs “no”, then we view r as an *invalid region*. Otherwise, we view r as a *candidate region* and then call our new Algorithm 4 (EXACT for short) on input (D_r, N_r, H_r, d). If EXACT outputs “yes”, then we view r as a *valid region*.

Let CR (respectively, VR) denote the set of candidate (respectively, valid) regions found by APPROX (respectively, EXACT). Then we process CR (respectively, VR) as follows: If there are two adjacent regions r_1 and r_2 (i.e., the finishing SNP site of r_1 is the same as the starting SNP site of r_2, or vice versa) on the chromosome C, we modify CR (respectively, VR) by merging r_1 and r_2 into a single region. In this way, we obtain two new sets CR' and VR' of regions. By experiments on some simulated data, we have found that VR' may miss some true mutation regions due to the errors added by the χ^2 model when generating the input genotype data. Therefore, we further modify VR' as follows: For a length-500 region $r \in CR \setminus VR$, if there are two regions r_1 and r_2 in VR' connected by r and the length ratio between r_1 and r_2 is between 0.2 and 5, we further modify VR' by replacing r_1 and r_2 with the smallest region that contains r_1 and r_2. In this way, we obtain a new set VR'' of regions. Finally, we output the first few longest regions in CR' and VR'' as the mutation regions of C. For convenience, we denote the heuristic producing CR' (respectively, VR'') by APPROX-HEU (respectively, EXACT-HEU).

The datasets used here to compare the performance of APPROX-HEU and EXACT-HEU are almost the same as those in [12]. The only difference is in the generation of the reference haplotype database H_r. Here, instead of deleting the haplotype data of the founders, we fix an error ratio ER (say, 5 %) and modify each

Table 1 The average *precision* and *recall* for P_2 through P_4, where each average is taken over 50 tests. The table consists of *two parts* separated by *two consecutive horizontal lines*. The *first* (respectively, *second*) *part* shows the results of APPROX-HEU (respectively, EXACT-HEU)

Heuristic	Pedigree					
	P_2		P_3		P_4	
	precision	*recall*	*precision*	*recall*	*precision*	*recall*
Longest	37.67 %	54.37 %	52.48 %	77.38 %	69.86 %	98.12 %
First-2-longest	36.66 %	85.98 %	47.33 %	92.59 %	62.08 %	98.59 %
First-3-longest	34.61 %	97.83 %	45.74 %	98.59 %	61.38 %	98.59 %
Longest	37.90 %	54.27 %	54.97 %	77.52 %	69.88 %	95.88 %
First-2-longest	37.78 %	87.85 %	48.97 %	90.18 %	63.28 %	96.34 %
First-3-longest	35.20 %	97.72 %	48.36 %	98.13 %	63.28 %	98.34 %

length-500 region r of the haplotype of each founder by flipping the bits at randomly selected $\lfloor 500 \cdot ER \rfloor$ positions of r and further put the modified haplotype back into H_r. In this way, we can make sure that if r is a true mutation region and the children's haplotype segments inherited from their parents are identical to those of their parents, then there must be a solution to $(D_r, N_r, H_r, \lfloor 500 \cdot ER \rfloor)$.

To compare the performance of APPROX-HEU and EXACT-HEU, we use the three different pedigrees P_2, P_3, and P_4 in [12]. P_2, P_3, and P_4 all have five generations but have 3, 4, and 5 diseased individuals in the youngest generation as part of the input to the programs, respectively.

We have done 50 experiments for each pedigree and calculated the average performance of the programs. The criteria used in the comparison are *precision* and *recall*. Recall that the *correctly detected mutation regions* by a program are the intersection of the regions outputted by the program and the true mutation regions. Here, *precision* is defined as the number of SNPs in the correctly detected mutation regions divided by the total number of SNPs in the regions outputted by the program. Moreover, *recall* is defined as the number of SNPs in the correctly detected mutation regions divided by the total number of SNPs in the true mutation regions. So, if *recall* is 1, then all the SNPs in the true mutation regions have been outputted by the program. Similarly, if *precision* is 1, then all the SNPs reported by the program are in the true mutation regions.

The experimental results for $ER = 5$ % (and hence $d = \lfloor 500 \cdot ER \rfloor = 25$) are summarized in Table 1, where columns P_2, P_3, and P_4 show the results for pedigrees P_2, P_3, and P_4, respectively. The table consists of two parts separated by two consecutive horizontal lines, where the first (respectively, second) part shows the result of APPROX-HEU (respectively, EXACT-HEU). Moreover, each part has three rows: Longest, First-2-longest, and First-3-longest. Longest shows the result that our program just outputs the longest detected region, while First-2-longest (respectively, First-3-longest) shows the result that our program outputs the first two (respectively, three) longest detected regions as the output. In each case, if the outputted regions have no overlap with the true mutation regions, both *precision* and *recall* are treated as 0.

Table 2 The total number of regions correctly excluded by EXACT-HEU for P_2 through P_4, where each total is taken over 50 tests

Pedigree	#yes	#correct	percentage
P_2	175	155	88.57 %
P_3	130	106	81.54 %
P_4	69	52	75.36 %

Table 3 The average length of the true mutation regions and that of the regions detected by APPROX-HEU and EXACT-HEU for P_2 through P_4

Pedigree	P_2	P_3	P_4
true region	5140	4893	3831
Longest	8657	9118	6516
First-2-longest	13307	12298	7493
First-3-longest	15681	13290	7639
Longest	8568	8857	6397
First-2-longest	13145	11849	7272
First-3-longest	15374	12874	7429

As can be seen from columns P_2, P_3, and P_3 in Table 1, the average *precision* of EXACT-HEU is slightly better than that of APPROX-HEU while the *recall* remains very much the same. In other words, the improvement is small. This might be explained as follows. Recall that when APPROX outputs "no" for a length-500 region r, EXACT outputs "no" for r as well. On the other hand, when APPROX outputs "yes" for r, EXACT may output "yes" or "no" for r. Thus, EXACT may output fewer length-500 regions than APPROX. Therefore, one may expect that EXACT gives better *precision* but worse *recall*.

We have done 50 experiments for each of pedigrees P_2, P_3, and P_4. The number of length-500 regions for which APPROX outputs "yes" but EXACT outputs "no" is given in column #yes of Table 2. Column #correct in Table 2 gives the number of regions for which EXACT outputs "no" and these regions indeed do not belong to the true mutation regions. As can be seen from the table, 88.57 % of the length-500 regions eliminated by EXACT are correct for pedigree P_2.

We have also investigated the reason why EXACT fails to output "yes" for some true mutation regions and we found that the χ^2 model also adds errors to the children's haplotype segments inherited from their parents. Thus, it is possible that the SC problem may have no solution to a region for the exact radius d.

Table 3 consists of three parts separated by two consecutive horizontal lines. The first part shows the average lengths of the true mutation regions for P_2, P_3, and P_4, while the second (respectively, third) part shows the average lengths of regions detected by APPROX-HEU (respectively, EXACT-HEU). We can see from the table that the average lengths of regions detected by EXACT-HEU are always shorter than those detected by APPROX-HEU. In particular, as can be seen from Table 1, this advantage is achieved without decreasing *recall* for pedigree P_3 when both APPROX-HEU and EXACT-HEU only outputs the longest detected region.

Our experiments have been performed on a Windows-7 desktop PC with Intel(R) Core(TM) 2 CPU (2.40 GHz) and 4 GB memory. Table 4 lists the running time of

Table 4 The longest, average, and shortest running times (in seconds) of EXACT-HEU on a length-500 region with $ER = 5\ \%$ for P_2 through P_4 over 50 tests

Pedigree	P_2	P_3	P_3
longest time	2.458	2.469	2.219
average time	0.213	0.166	0.059
shortest time	0.006	0.007	0.009

Table 5 The longest, average, and shortest running times (in minutes) of EXACT-HEU on the whole chromosome with $ER = 5\ \%$ for P_2 through P_4 over 50 tests

Pedigree	P_2	P_3	P_3
longest time	41.950	17.500	7.600
average time	8.138	4.646	2.118
shortest time	1.167	0.583	0.167

EXACT-HEU for a length-500 region with $ER = 5\ \%$. We can see from the table that on average, it takes less than a second to investigate a single length-500 region. Moreover, Table 5 shows the total running time of EXACT-HEU for investigating a whole chromosome (Chromosome 1) with $ER = 5\ \%$.

6.1 Modifying APPROX-HEU and EXACT-HEU

In the aforementioned experiments with APPROX-HEU (respectively, EXACT-HEU), we investigate length-500 regions one by one using different center indices p. Therefore, APPROX-HEU (respectively, EXACT-HEU) can say "yes" for different length-500 regions based on different p's.

Suppose that we modify APPROX-HEU (respectively, EXACT-HEU) as follows. After finding all candidate (respectively, valid) length-500 regions in a whole chromosome, we choose the index p that appears the most among all the reported candidate (respectively, valid) length-500 regions and use this p to further verify all the reported candidate (respectively, valid) regions. In other words, we here use an extra condition that for a fixed index p (instead of different p's for different regions), the instance $(D_r, N_r, H_r, 2d)$ (respectively, (D_r, N_r, H_r, d)) of the SC problem for each candidate (respectively, valid) length-500 region r must have a solution where the center haplotype segment s is within a distance of at most $2d$ (respectively, d) from $h_p \in H_r$. For convenience, we denote the modified APPROX-HEU (respectively, EXACT-HEU) by APPROX-HEU$'$ (respectively, EXACT-HEU$'$).

We have done 50 tests with APPROX-HEU$'$ and EXACT-HEU$'$ for pedigrees P_2, P_3, and P_4 with different ER's. The results are summarized in Tables 6, 7, and 8. In general, EXACT-HEU$'$ achieves better *precision* than APPROX-HEU$'$, while they achieve almost the same *recall*.

6.2 Comparing the Speed of Algorithms 4 and 5

We also compare the average running time of Algorithms 4 and 5 on a length-500 region with $ER = 8\ \%$ for pedigree P_4, where each average is taken over 100 tests. The experimental results are summarized in Table 9 where we can see that Algorithm 4

Table 6 The average *precision* and *recall* achieved by APPROX-HEU$'$ and EXACT-HEU$'$ for P_2, where each percentage is taken over 50 tests. The table consists of *3 parts* separated by *two consecutive horizontal lines*. *The first* (respectively, *second*) *part* shows the results of APPROX-HEU$'$ (respectively, EXACT-HEU$'$), while *the third part* shows the average running time (in minutes) of EXACT-HEU$'$ on the whole chromosome

Heuristic	Error Ratio							
	5 %		6 %		7 %		8 %	
	precision	*recall*	*precision*	*recall*	*precision*	*recall*	*precision*	*recall*
Longest	29.06 %	40.37 %	38.76 %	51.47 %	38.13 %	52.78 %	38.44 %	40.61 %
First-2-longest	29.73 %	48.07 %	36.46 %	56.60 %	37.21 %	65.17 %	33.02 %	50.22 %
First-3-longest	28.84 %	49.30 %	36.39 %	61.32 %	36.60 %	69.61 %	31.60 %	57.02 %
Longest	31.89 %	43.84 %	40.84 %	53.03 %	38.91 %	56.53 %	37.05 %	52.30 %
First-2-longest	32.25 %	50.2 %	37.99 %	56.13 %	38.86 %	65.62 %	33.96 %	56.28 %
First-3-longest	31.91 %	50.83 %	38.17 %	57.67 %	38.28 %	66.78 %	32.83 %	57.23 %
running time	34.732		37.691		49.971		71.181	

Table 7 The average *precision* and *recall* achieved by APPROX-HEU$'$ and EXACT-HEU$'$ for P_3, where each percentage is taken over 50 tests. The table consists of *3 parts* separated by *two consecutive horizontal lines*. *The first* (respectively, *second*) *part* shows the results of APPROX-HEU$'$ (respectively, EXACT-HEU$'$), while *the third part* shows the average running time (in minutes) of EXACT-HEU$'$ on the whole chromosome

Heuristic	Error Ratio							
	5 %		6 %		7 %		8 %	
	precision	*recall*	*precision*	*recall*	*precision*	*recall*	*precision*	*recall*
Longest	57.75 %	67.55 %	60.57 %	68.45 %	56.62 %	72.17 %	39.44 %	59.79 %
First-2-longest	52.29 %	69.51 %	57.14 %	74.30 %	53.58 %	75.03 %	40.05 %	68.98 %
First-3-longest	49.77 %	70.64 %	55.50 %	75.78 %	52.95 %	77.21 %	38.72 %	71.62 %
Longest	61.10 %	73.35 %	62.86 %	72.40 %	56.82 %	73.44 %	47.99 %	69.95 %
First-2-longest	59.06 %	74.71 %	61.72 %	77.85 %	56.21 %	75.81 %	45.19 %	70.14 %
First-3-longest	57.89 %	74.71 %	61.11 %	77.85 %	55.81 %	75.81 %	44.96 %	70.63 %
running time	24.217		26.341		33.861		48.649	

is faster than Algorithm 5. It is worth mentioning that although $d = 500 \cdot ER = 40$ in our experiments, $d - d_p$ is usually much smaller than d.

To compare the speed of Algorithms 4 and 5, we have also run the programs on other simulated datasets. To generate a simulated dataset, we here generate an instance $(s_1, \ldots, s_k, L, d)$ of the BCSS problem and then use it to obtain an instance of the SC problem via the reduction in the proof of Theorem 4.1. Each instance $(s_1, \ldots, s_k, L, d)$ of the BCSS problem is generated as follows.

1. Generate k random binary strings $s_1, \ldots, s_k$ of the same length K, and also generate a random binary string t of length L.

Table 8 The average *precision* and *recall* achieved by APPROX-HEU′ and EXACT-HEU′ for P_4, where each percentage is taken over 50 tests. The table consists of *3 parts* separated by *two consecutive horizontal lines*. *The first* (respectively, *second*) *part* shows the results of APPROX-HEU′ (respectively, EXACT-HEU′), while *the third part* shows the average running time (in minutes) of EXACT-HEU′ on the whole chromosome

Heuristic	Error Ratio							
	5 %		6 %		7 %		8 %	
	precision	*recall*	*precision*	*recall*	*precision*	*recall*	*precision*	*recall*
Longest	67.53 %	97.98 %	67.53 %	97.23 %	71.02 %	91.33 %	76.78 %	94.94 %
First-2-longest	63.87 %	97.98 %	64.31 %	98.68 %	68.21 %	93.25 %	70.99 %	95.85 %
First-3-longest	63.69 %	97.98 %	64.00 %	98.68 %	67.75 %	93.94 %	69.47 %	95.85 %
Longest	68.34 %	96.71 %	67.96 %	96.97 %	71.22 %	92.25 %	74.96 %	94.79 %
First-2-longest	65.85 %	98.88 %	65.28 %	97.95 %	70.03 %	93.04 %	72.43 %	95.67 %
First-3-longest	65.85 %	98.88 %	65.12 %	97.95 %	70.10 %	93.27 %	72.46 %	95.88 %
running time	10.602		10.641		18.327		22.280	

Table 9 The average running time (in seconds) of Algorithms 4 and 5 for simulated biological data, where k is the number of diseased individuals and d is the radius

"yes" instances				"no" instances			
d	k	Algorithm 4	Algorithm 5	d	k	Algorithm 4	Algorithm 5
40	5	0.028	0.680	40	5	0.142	0.203

2. For each $i \in \{1, \ldots, k\}$, first obtain a string t_i from t by selecting d positions uniformly at random and flipping the bits of t at the selected positions, and further modify s_i by first selecting a position $q \in \{1, \ldots, K - L + 1\}$ uniformly at random and then replacing the substring $s_i[q..q + L - 1]$ with t_i.

There are three merits of generating $(s_1, \ldots, s_k, L, d)$ in the above way. First, $(s_1, \ldots, s_k, L, d)$ always has a solution but $(s_1, \ldots, s_k, L, d - 1)$ usually does not. Secondly, $d - d_p$ is almost always equal to d. Thirdly, it is easy to choose a triple (k, L, d) such that it takes time to solve each of $(s_1, \ldots, s_k, L, d)$ and $(s_1, \ldots, s_k, L, d - 1)$. Because of the three merits, we can obtain a meaningful comparison in speed between Algorithms 4 and 5 for both "yes" and "no" instances. In our experiments, we fix $K = 75$ and $L = 60$ but choose k from $\{10, 15, 20\}$ and d from $\{10, 11, 12\}$. These choices are made so that the generated instances of the BCSS problem are reduced to instances of the SC problem that are close to the simulated biological datasets used in our aforementioned linkage-analysis experiments. For each combination of (K, L, k, d), we generate 20 random instances and summarize the average running time (in seconds) of Algorithms 4 and 5 in Table 10. By the table, Algorithm 4 seems to be always faster than Algorithm 5. The superiority of Algorithm 4 over Algorithm 5 becomes more obvious when d becomes larger.

Table 10 The average running time (in seconds) of Algorithms 4 and 5 for simulated datasets, where k is the number of diseased individuals and d is the radius

"yes" instances				"no" instances			
d	k	Algorithm 4	Algorithm 5	d	k	Algorithm 4	Algorithm 5
10	10	0.184	2.415	9	10	0.341	1.592
11	10	2.430	21.106	10	10	1.017	7.508
12	10	19.403	164.523	11	10	5.214	47.002
10	15	0.616	6.562	9	15	1.045	3.069
11	15	2.341	29.724	10	15	1.831	12.596
12	15	21.111	238.538	11	15	5.011	62.805
10	20	0.449	7.277	9	20	2.279	4.778
11	20	2.402	43.508	10	20	2.887	17.196
12	20	20.203	381.833	11	20	7.052	94.158

6.3 Conclusion

In summary, an exact algorithm for the SC problem can be used to achieve usually better *precision* and keep *recall* much the same. Honestly speaking, the improvement in *precision* does not look so significant. Therefore, it remains an open problem how to make use of an exact algorithm for the SC problem in linkage analysis.

Acknowledgements We thank the referees for very helpful comments. This work is supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 121608).

References

1. Cai, Z., Sabaa, H., Wang, Y., Goebel, R., Wang, Z., Xu, J., Stothard, P., Lin, G.: Most parsimonious haplotype allele sharing determination. BMC Bioinform. **10**, 115 (2009)
2. Chen, Z.-Z., Wang, L.: Fast exact algorithms for the closest string and substring problems with application to the planted (L, d)-motif model. IEEE/ACM Trans. Comput. Biol. Bioinform. **8**(5), 1400–1410 (2011)
3. Chen, Z.-Z., Ma, B., Wang, L.: A three-string approach to the closest string problem. J. Comput. Syst. Sci. **78**, 164–178 (2012)
4. Doi, K., Li, J., Jiang, T.: Minimum recombinant haplotype configuration on tree pedigrees. In: Proceedings of Workshop on Algorithms in Bioinformatics (WABI), pp. 339–353 (2003)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monogr. Comput. Sci. Springer, New York (1999)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
7. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica **37**, 25–42 (2003)
8. Impagliazzo, R., Paturi, R.: The Complexity of k-SAT. In: Proceedings of the 14th IEEE Conference on Computational Complexity, pp. 237–240 (1999)
9. Li, J., Jiang, T.: An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. In: Proceedings of Symposium on Computational Molecular Biology (RECOMB), pp. 20–29 (2004)
10. Li, J., Jiang, T.: Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. J. Comput. Biol. **12**(6), 719–739 (2005)

11. Lin, G., Wang, Z., Wang, L., Lau, Y.-L., Yang, W.: Identification of linked regions using high-density SNP genotype data in linkage analysis. Bioinformatics **24**(1), 86–93 (2008)
12. Ma, W., Yang, Y., Chen, Z.-Z., Wang, L.: Mutation region detection for closely related individuals without a known pedigree. IEEE/ACM Trans. Comput. Biol. Bioinform. **9**(2), 499–510 (2012)
13. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. SIAM J. Comput. **39**, 1432–1443 (2009)
14. Marx, D.: Closest substring problems with small distances. SIAM J. Comput. **38**, 1382–1410 (2008)
15. Qian, D., Beckmann, L.: Minimum recombinant haplotyping in pedigrees. Am. J. Hum. Genet. **70**, 1434–1445 (2002)
16. Tapadar, P., Ghosh, S., Majumder, P.P.: Haplotyping in pedigrees via a genetic algorithm. Hum. Hered. **43**, 56 (1999)
17. Wang, L., Zhu, B.: Efficient algorithms for the closest string and distinguishing string selection problems. In: Proceedings of the 3rd International Workshop on Frontiers in Algorithmics, pp. 261–270 (2009)
18. Xiao, J., Liu, L., Xia, L., Jiang, T.: Fast elimination of redundant linear equations and reconstruction of recombination-free Mendelian inheritance on a pedigree. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 655–664 (2007)
19. Zhang, K., Sun, F., Zhao, H.: Haplore: a program for haplotype reconstruction in general pedigrees without recombination. Bioinformatics **21**, 90–103 (2005)
20. Zhao, R., Zhang, N.: A more efficient closest string algorithm. In: Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (BICoB), pp. 210–215 (2010)