Identifying Maximal Perfect Haplotype Blocks

Luís Cunha[1,2][0000−0002−3797−6053], Yoan Diekmann[3][0000−0003−0030−0786], Luis Kowada[1][0000−0002−7975−0060], and Jens Stoye[1,4][0000−0002−4656−7155]

[1] Universidade Federal Fluminense, Niterói, Brazil
[2] Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil
[3] Department of Genetics, Evolution and Environment, University College London, London WC1E 6BT, UK
[4] Faculty of Technology and Center for Biotechnology, Bielefeld University, Bielefeld, Germany

Abstract. The concept of maximal perfect haplotype blocks is introduced as a simple pattern allowing to identify genomic regions that show signatures of natural selection. The model is formally defined and a simple algorithm is presented to find all perfect haplotype blocks in a set of phased chromosome sequences. Application to three whole chromosomes from the 1000 genomes project phase 3 data set shows the potential of the concept as an effective approach for quick detection of selection in large sets of thousands of genomes.

Keywords: Population genomics · selection coefficient · haplotype block.

1 Introduction

Full genome sequences are amassing at a staggering yet further accelerating pace. For humans, multiple projects now aim to deliver numbers five orders of magnitudes higher than the initial human genome project 20 years ago[5]. This development is fuelled by drastic reductions in sequencing costs, by far exceeding Moore's law [6]. As a result, the bottleneck in genomics is shifting from data production to analysis, calling for more efficient algorithms scaling up to ever-larger problems.

A simple yet highly interesting pattern in population genomic datasets are fully conserved haplotype blocks (called *maximal perfect haplotype blocks* in the following). When large and frequent enough in the population, they may be indicative for example of a selective sweep. Their simple structure simplifies analytical treatment in a population genetic framework. To our surprise, we could not locate any software tool that can find them efficiently.

In this paper, we present a simple and efficient algorithm for finding all maximal perfect haplotype blocks in a set of binary sequences. While the algorithm is not optimal in terms of worst-case analysis, and therefore this paper ends

[5] *E.g.*, https://www.genomicsengland.co.uk/the-100000-genomes-project-by-numbers

with an interesting open question, it has the convenient property allowing its *lazy* implementation, making it applicable to large real data sets in practice.

The paper is organized as follows: The problem of finding all maximal perfect haplotype blocks in a set of binary sequences is formally defined in Section 2. The trie data structure we use and our algorithm are presented in Section 3. In Section 4 we describe our lazy implementation of the algorithm and show its applicability to real data. Section 5 concludes with an open problem and two ideas for alternative algorithmic approaches.

2 Basic Definitions

The input data to our problem are k binary sequences, all of the same length n, each one representing a chromosome. Each column refers to a biallelic[6] single nucleotide polymorphism (SNP), with entries 0 and 1 corresponding to different but otherwise arbitrary alleles, although polarised data (where 0 and 1 refer to ancestral and derived allele, respectively) usually helps the interpretation of results. Of special interest are large blocks of conservation, that are stretches of identical alleles present at the same loci in many of the input sequences. Formally, we define such blocks as follows.

Definition 1. *Given an ordered set* $X = (x_1, \ldots, x_k)$ *and an index set* $I = \{i_1, \ldots, i_\ell\}$, $0 \leq \ell \leq k$, $1 \leq i_j \leq k$, *the* I-induced subset of X *is the set* $X|_I = \{x_{i_1}, \ldots, x_{i_\ell}\}$.

Definition 2. *Given* k *binary sequences* $S = (s_1, \ldots, s_k)$, *each of length* n, *a* maximal haplotype block *is a triple* (K, i, j) *with* $K \subseteq \{1, \ldots, k\}$, $|K| \geq 2$ *and* $1 \leq i \leq j \leq n$ *such that*

1. $s[i..j] = t[i..j]$ *for all* $s, t \in S|_K$,
2. $i = 1$ *or* $s[i-1] \neq t[i-1]$ *for some* $s, t \in S|_K$ (left-maximality),
3. $j = n$ *or* $s[j+1] \neq t[j+1]$ *for some* $s, t \in S|_K$ (right-maximality), *and*
4. *there exists no* $K' \subseteq \{1, \ldots, k\}$, $K' \supsetneq K$, *such that* $s[i..j] = t[i..j]$ *for all* $s, t \in S|_{K'}$.

The formal problem we address in this paper can then be phrased as follows.

Problem 1. Given k binary sequences $S = (s_1, \ldots, s_k)$, each of length n, find all maximal haplotype blocks in S.

The following proposition gives a simple upper bound for the output of this problem.

[6] For convenience, we exclude multiallelic sites which may contain alleles coded as 2 or 3, or merge the minor alleles if they are rare and represent them as 1. These make up only a small fraction of the total SNPs in real data, and we therefore do not expect any overall effect.

Proposition 1. *Given k binary sequences $S = (s_1, \ldots, s_k)$, each of length n, there can be only $O(kn)$ maximal haplotype blocks in S.*

Proof. We argue that at any position i, $1 \leq i \leq n$, there can start at most $k - 1$ maximal haplotype blocks. This follows from the maximality condition and the fact that a maximal haplotype block contains at least two sequences whose longest common prefix it is. □

As the following example shows, for sufficiently large n the bound given in Proposition 1 is tight.

Example 1. Consider the family of sequences $S_{k,n} = (s_1, s_2, \ldots, s_k)$, each of length n, defined as follows: $s_1 = 0^n$, $s_2 = 1^n$, followed by *chunks* of sequences $c_1, c_2, \ldots$ such that chunk c_i contains $2i$ sequences, that are all sequences of length n which are repetitions of $0^i 1^i$ and its rotations. The last chunk may be truncated, so that the total number of sequences is k. For example, for $k = 14$ and $n = 24$ we have:

$$\begin{aligned}
s_1 &= 000000000000000000000000 \\
s_2 &= 111111111111111111111111 \\
&\left.\begin{aligned}
s_3 &= 010101010101010101010101 \\
s_4 &= 101010101010101010101010
\end{aligned}\right\} c_1 \\
&\left.\begin{aligned}
s_5 &= 001100110011001100110011 \\
s_6 &= 011001100110011001100110 \\
s_7 &= 110011001100110011001100 \\
s_8 &= 100110011001100110011001
\end{aligned}\right\} c_2 \\
&\left.\begin{aligned}
s_9 &= 000111000111000111000111 \\
s_{10} &= 001110001110001110001110 \\
s_{11} &= 011100011100011100011100 \\
s_{12} &= 111000111000111000111000 \\
s_{13} &= 110001110001110001110001 \\
s_{14} &= 100011100011100011100011
\end{aligned}\right\} c_3
\end{aligned}$$

Analysis shows that in chunk c_i, $i \geq 1$, every substring of length i forms a new maximal haplotype block together with some substring in one of the earlier sequences. Therefore, for $n > k$ the total number of maximal haplotype blocks in $S_{k,n}$ grows as $\Theta(kn)$.

3 Algorithm

Our algorithm to find all maximal haplotype blocks in a set of sequences S uses the (binary) trie of the sequences in S. For completeness, we recall its definition:

Definition 3. *The* trie *of a set of sequences S over an alphabet Σ is the rooted tree whose edges are labeled with characters from Σ and the labels of all edges starting at the same node are distinct, such that the concatenated edge labels from the root to the leaves spell exactly the sequences in S.*

A *branching vertex* in a rooted tree is a vertex with out-degree larger than one.

Example 2. Figure 1 shows the trie $T_1(S)$ of $k = 4$ binary strings of length $n = 6$. It has three branching vertices.

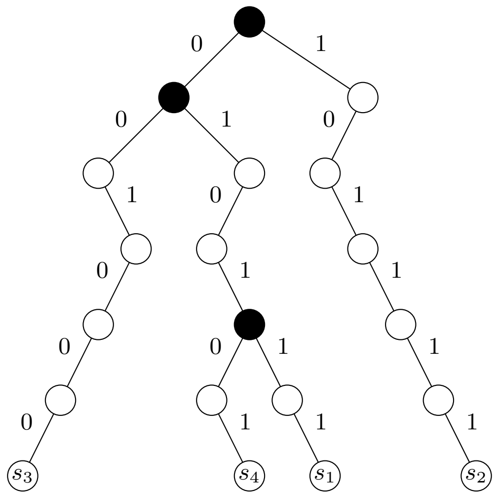

Fig. 1. Trie $T_1(S)$ of $k = 4$ binary strings $S = (s_1, s_2, s_3, s_4)$ with $s_1 = 010111$, $s_2 = 101111$, $s_3 = 001000$ and $s_4 = 010101$. Branching vertices are indicated by filled nodes.

It is well known that the trie of a set of sequences S over a constant-size alphabet can be constructed in linear time and uses linear space with respect to the total length of all sequences in S.

Our algorithm to find all maximal haplotype blocks of k binary sequences S, each of length n, iteratively constructs the trie of the suffixes of the sequences in S starting at a certain index i, $i = 1, 2, \ldots, n$. We denote the ith trie in this series of tries by $T_i(S)$.

Observation 1 *All branching vertices of $T_1(S)$ correspond to maximal haplotype blocks starting at index 1 of sequences in S.*

This follows from the fact that a branching vertex in $T_1(S)$ corresponds to a common prefix of at least two sequences in S that are followed by two different characters, thus they are right-maximal. Left-maximality is automatically given since $i = 1$.

Example 2 (cont'd). The tree $T_1(S)$ in Figure 1 has three branching vertices, corresponding to the maximal haplotype blocks starting at index 1: the string

0101 occurring as a prefix in sequences s_1 and s_4; the string 0 occurring as a prefix in sequences s_1, s_3 and s_4; and the empty string (at the root of the tree) occurring as a prefix of all four strings.

In order to find maximal haplotype blocks that start at later positions $i > 1$, essentially the same idea can be used, just based on the tree $T_i(S)$. The only difference is that, in addition, one needs explicitly to test for left-maximality. As the following observation shows, this is possible by looking at the two subtrees of the root of the previous trie, $T_{i-1}(S)$.

Observation 2 *A haplotype block starting at position $i > 1$ is left-maximal if and only if it contains sequences that are in the* 0*-subtree of the root of* $T_{i-1}(S)$ *and sequences that are in the* 1*-subtree of the root of* $T_{i-1}(S)$.

Example 2 (cont'd). As shown in Figure 2, trie $T_2(S)$ has three branching vertices, corresponding to the right-maximal haplotype blocks starting at index 2: the string 101 occurring at positions 2..4 in sequences s_1 and s_4; the string 01 occurring at positions 2..3 in sequences s_2 and s_3; and, again, the empty string. The string 101 is not left-maximal (and therefore not maximal), visible from the fact that s_1 and s_4 were both in the same (0-) subtree of the root in $S_1(T)$. The other two right-maximal haplotype blocks are also left-maximal.

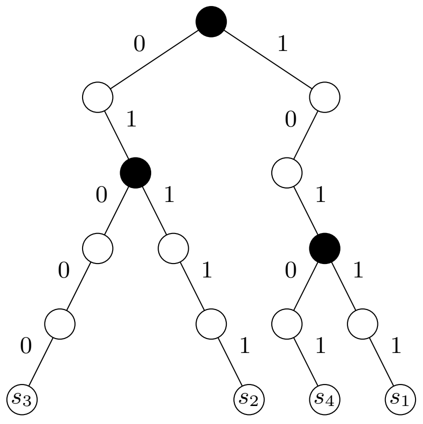

Fig. 2. Trie $T_2(S)$ for the strings from Figure 1.

The algorithm to find all maximal haplotype blocks in a set of k sequences S, each of length n, follows immediately. It first constructs the trie $T_1(S)$ and locates all prefix haplotype blocks by a simple depth-first traversal. Then, iteratively for $i = 2, 3, \ldots, n$, $T_i(S)$ is constructed by merging the 1-subtree into the 0-subtree of the root of $T_{i-1}(S)$ during a parallel traversal of the two sister-subtrees. The former 0-subtree will then be $T_i(S)$. While doing so, branching vertices with leaves that came from both of these subtrees are reported as maximal blocks starting from index i. Pseudocode is given in Algorithm 1.

Algorithm 1 (Haploblocks)

```
Input: $k$ binary sequences $S = (s_1, \ldots, s_k)$, each of length $n$
Output: all maximal haplotype blocks of $S$
 1: construct $T \leftarrow T_1(S)$
 2: for each branching vertex $v$ of $T$ do
 3:     report maximal block at positions $1..d-1$, where $d$ is the depth of $v$ in $T$
 4: end for
 5: for $i = 2, \ldots, n$ do
 6:     merge-and-report($T$.left, $T$.right; $i, 0$)
 7:     $T \leftarrow T$.left
 8: end for

Function merge-and-report($l, r; i, d$)
 9: if $l$ is empty then
10:     $l \leftarrow r$    ▷ simplification of presentation: implemented through call by reference
11:     return
12: end if
13: if $r$ is empty then
14:     return
15: end if
16: leftmaximal $\leftarrow$ not ($l$.left and $r$.left are empty or $l$.right and $r$.right are empty)
17: if $l$.left is empty then
18:     $l$.left $\leftarrow r$.left
19: else
20:     merge-and-report($l$.left, $r$.left; $i, d+1$)
21: end if
22: if $l$.right is empty then
23:     $l$.right $\leftarrow r$.right
24: else
25:     merge-and-report($l$.right, $r$.right; $i, d+1$)
26: end if
27: rightmaximal $\leftarrow$ not ($l$.left is empty or $l$.right is empty)
28: if leftmaximal and rightmaximal then
29:     report maximal block at positions $i..i+d$
30: end if
```

Analysis. The overall running time of Algorithm 1 is $O(kn^2)$. This can be seen easily as follows. The initial construction of $T_1(S)$ takes linear time in the input size, thus $O(kn)$ time. Similar for the identification of maximal haplotype blocks starting at index $i = 1$. Each of the following $n-1$ iterations performs in the worst case a traversal of the complete tree that has size $O(kn)$, thus taking $O(kn^2)$ time in total.

Note that, as presented in the pseudocode of Algorithm 1, the algorithm only reports the start and end positions i and j, respectively, of a maximal haplotype block (K, i, j), but not the set of sequences K where the block occurs. This can easily be added if, whenever in lines 3 and 29 some output is generated, the current subtree is traversed and the indices of all $|K|$ sequences found at the leaves are collected and reported. Such a traversal, however, costs $O(n \cdot |K|)$ time in the worst case, resulting in an overall running time of $O(kn^2 + n \cdot |\text{output}|)$. An alternative could be to store at each branching vertex of the trie as *witness* the index of a single sequence in the subtree below. This would allow to report, in addition to start and end positions i and j, respectively, also the sequence of a maximal haplotype block (K, i, j). If desired, the set K can then be generated easily using standard pattern matching techniques on the corresponding intervals of the k input sequences in $O(k \cdot (j - i))$ time.

4 Results

4.1 Data

To evaluate our algorithm, we downloaded chromosomes 2, 6 and 22 of the 1000 genomes phase 3 data set, which provides phased whole-genome sequences of 2504 individuals from multiple populations world-wide [1]. We extracted biallelic SNPs and represented the data as a binary matrix with help of the *cyvcf2* Python library [10].

4.2 Our Implementation of Algorithm 1

We implemented Algorithm 1 in C. Thereby we encountered two practical problems.

First, the recursive structure of Algorithm 1, when applied to haplotype sequences that are several hundred thousand characters long, produces a program stack overflow. Therefore we re-implemented the tree construction and traversal in a non-recursive fashion, using standard techniques as described, e.g., on the "Non-recursive depth first search algorithm" page of Stack Overflow[7].

Second, the constructed trie data structure requires prohibitive space. For example, already for the relatively small chromosome 22, $T_1(S)$ has 5,285,713,633 vertices and thus requires (in our implementation with 32 bytes per vertex) more than 157 gigabytes of main memory. However, most of the vertices are in non-branching paths to the leaves, corresponding to unique suffixes of sequences in S. Since such paths can never contain a branching vertex, they are not relevant. They become of interest only later in the procedure when the path is merged with other paths sharing a common prefix. Therefore we implemented a *lazy* version of our data structure, that stops the construction of a path whenever it contains only a single sequence. During the merge-and-report procedure, then, whenever an unevaluated part of the tree is encountered, the path has to be

[7] https://stackoverflow.com

extended until it branches and paths represent single sequences again. This has the effect that at any time only the top part of the trie is explicitly stored in memory. For chromosome 22, the maximum number of nodes that are explicitly stored at once drops to 5,677,984, reducing the memory footprint by about a factor of 1,000. In fact, this number is not much larger for any other of the human chromosomes that we tested, since it depends on the size of the maximal perfect haplotype blocks present in the data, and not on the chromosome length.

Table 1 contains memory usage and running times for all three human chromosomes that we studied. All computations were performed on a Dell RX815 machine with 64 2.3 GHz AMD Opteron processors and 512 GB of shared memory.

Table 1. Resources used by our implementation of Algorithm 1 when applied to the three data sets described in Section 4.1.

data set	length	memory	time
chr. 2	6,786,300	33.67 GB	2h 37min
chr. 6	4,800,101	23.91 GB	1h 51min
chr. 22	1,055,454	5.45 GB	25min

4.3 Interpretation of Results

In order to demonstrate the usefulness of the concept of haplotype blocks and our algorithm and implementation to enumerate them, we show how our results can form the efficient algorithmic core of a genome-wide selection scan.

Given a maximal perfect haplotype block (K, i, j) found in a set of k chromosomes, we estimate the selection coefficient s and the time t since the onset of selection following the approach presented by Chen *et al.* [3]. Therefore, we first convert the physical positions corresponding to indices i and j of the block from base pairs to a genetic distance d quantifying genetic linkage in centimorgan[8], which is the chromosomal distance for which the expected number of crossovers in a single generation is 0.01. Distance value d in turn is converted to the recombination fraction r – defined as the ratio of the number of recombined gametes between two chromosomal positions to the total number of gametes produced – using Haldane's map function

$$r = \frac{1 - \exp(-\frac{2d}{100})}{2}. \tag{1}$$

With r, K, k we can define a likelihood function $\mathcal{L}(s \,|\, r, K, k)$ allowing to compute maximum likelihood estimates of the selection coefficient and time since the onset

[8] A genetic map required to do so is available for example as part of Browning *et al.* [2] at http://bochet.gcc.biostat.washington.edu/beagle/genetic_maps.

of selection, $\hat{s}$ and $\hat{t}$, respectively. The full derivation from population genetic theory is outside the scope of this paper, and the subsequent paragraphs merely intend to provide some basic intuition. For more details, we refer the interested reader to Chen *et al.* [3] and the appendix of reference [4].

First, assume a deterministic model for the frequency change of an allele with selective advantage s, which yields a sigmoidal function over continuous time t that ignores the stochasticity in frequency trajectories for small s,

$$y_t \;=\; \frac{y_0}{y_0 + (1-y_0)e^{-st}}, \tag{2}$$

where y_0 is the initial allele frequency at the onset of selection[9] and $y_t = \frac{|K|}{k}$ is the observed allele frequency assumed to be representative of the population frequency. Equation 2 links up the selection coefficient s and the age t of the allele, for example requiring larger selective advantage to reach a given frequency if the allele is young.

Next, we exploit the fact that the recombination rate is independent of selection, and if assumed to be constant through time can therefore be seen to behave as a "recombination clock". Given a haplotype of length such that its recombination fraction is r, moreover, with an allele at one end that at time t segregates at frequency $y(t)$ in the population, the expected number of recombination events C altering that allele in the time interval $[0, t]$ can be obtained by

$$C \;=\; r\int_{u=0}^{t} (1 - y(u))\,du \;=\; r\left(t - \frac{1}{s}\ln(1 - y_0 + e^{st}y_0)\right), \tag{3}$$

where the second equality follows from substituting in Equation 2. Assuming that the number of recombination events follows a Poisson distribution, the probability of no event, *i.e.* of full conservation of a haplotype after time t, becomes

$$e^{-C} \;=\; e^{-rt}(1 - y_0(1 - e^{st}))^{\frac{r}{s}}. \tag{4}$$

Finally, one can define the likelihood of observing a haplotype block for a given s and t as $|K|$ times the probability of a conserved haplotype (Equation 4) times the probability of recombination events at the borders (Equation 3). As usual, the logarithm simplifies the equation, yielding

$$\ln \mathcal{L}(s|r, K, k) \propto$$
$$-rt + \frac{r}{s}\ln(1 - y_0(1 - e^{st})) + \ln\left(t - \frac{1}{s}\ln(1 - y_0(1 - e^{st}))\right), \tag{5}$$

with t being directly derived from Equation 2:

$$t = \frac{1}{s}\ln\left(\frac{y_t(1 - y_0)}{y_0(1 - y_t)}\right). \tag{6}$$

[9] In the following, y_0 is arbitrarily fixed at 0.00005, corresponding to $\frac{1}{2N_e}$ with an effective population size N_e = 10,000.

Note that we omit the factor $|K|$ and summands $\ln(r)$ for the recombination fractions at the borders of the haplotype (see Equation 3) that we assume are small and approximately equal, as they are inconsequential for optimization. Also, the massive speed gain of our approach trades off with a systematic but conservative underestimation of $\hat{s}$ when compared to the original equation in reference [3] as we do not consider the full varying extent of the individual haplotypes.

Equation 5 can be evaluated for a range of values to find the (approximate) maximum likelihood estimate $\hat{s}$ at a given precision, *e.g.* $s \in \{0.001, 0.002, \dots\}$ to estimate $\hat{s}$ with error below 0.001. Once $\hat{s}$ has been found, the corresponding time $\hat{t}$ is obtained by substituting $\hat{s}$ into Equation 6.

As the Haploblocks algorithm is able to rapidly scan entire chromosomes, and estimating $\hat{s}$ and $\hat{t}$ requires to evaluate only simple analytical expressions, one can efficiently generate a genome-wide selection track. Figure 3 illustrates the results for the locus known to contain one of the strongest signals of selection detected so far, the lactase persistence allele in modern Europeans -13.910:C>T (rs4988235). The selection coefficient we compute is consistent with the range of current estimates (see Ségurel and Bon [11] and references therein).

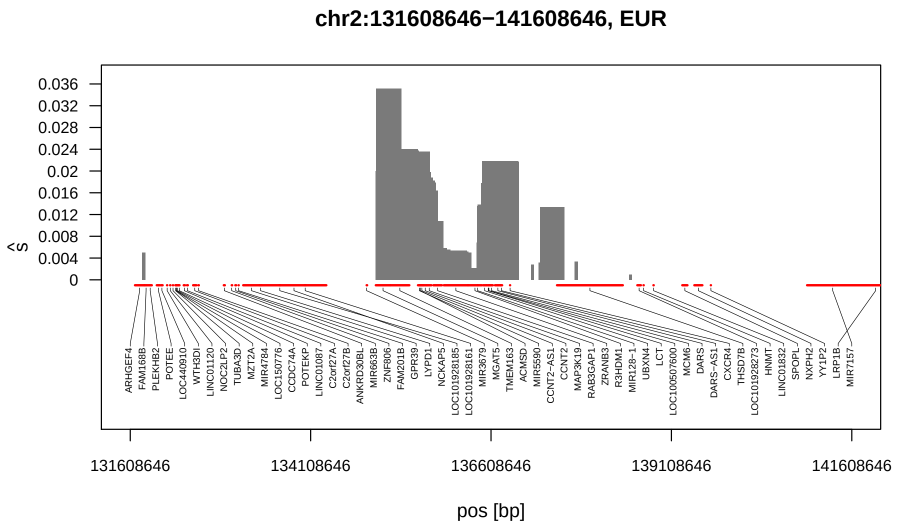

Fig. 3. Maximum likelihood estimates of selection coefficients for the locus containing the lactase gene in European individuals from the 1000 genomes data set. Each block of weight above 500,000 was converted to a selection coefficient applying Equation 5 on a set of values $\{0.0002, 0.0004, \dots\}$ and choosing the (approximate) maximum likelihood estimate $\hat{s}$. Red lines indicate genes annotated in the RefSeq database [9].

5 Conclusion

We presented an $O(kn^2)$ time algorithm for finding all maximal perfect haplotype blocks in a set of k binary sequences, each of length n, that scales well in practice. Even large human chromosomes can be processed in a few hours using moderate amounts of memory.

This allowed us to design an analytical approach with enumeration of maximal perfect haplotype blocks at its core that not only detects selection genome-wide efficiently, but does so by directly estimating a meaningful and interpretable parameter, the selection coefficient s.

As a proof of principle, we applied our method and evaluated the results for a locus known to contain one of the strongest signals of selection detected so far, and obtained a value for s consistent with current estimates.

It remains an open question if there exists an optimal algorithm for finding all maximal perfect haplotype blocks, *i.e.*, an algorithm that runs in $O(kn)$ time.

It could be worthwhile to study the bipartite graph $(U \cup W, E)$ in which the vertices in $U = \{u_1, \ldots, u_k\}$ correspond to the sequences in S and the vertices in $W = \{w_1, \ldots, w_n\}$ to index positions in these sequences. An edge $(u_i, w_j) \in E$ is drawn if and only if $s_i[j] = 1$. Problem 1 is then equivalent to finding all twin vertices (sets of vertices with identical neighborhood) in intervals of vertices in W. Figure 4 shows this graph for the sequences from Example 2.

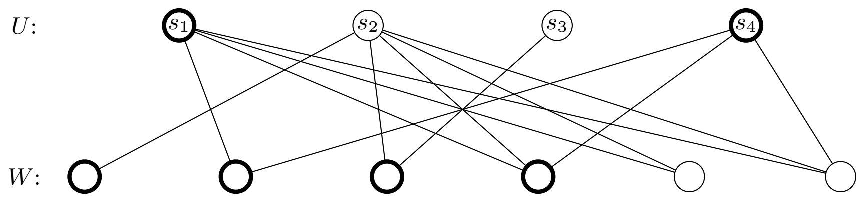

Fig. 4. Bipartite graph $(U \cup W, E)$ representing the four binary sequences $s_1 = 010111$, $s_2 = 101111$, $s_3 = 001000$ and $s_4 = 010101$. Haplotype blocks can be identified as sets of twins when the vertices in the lower row W are restricted to a consecutive interval. For example, s_1 and s_4 are twins in the interval formed by the first four vertices of W (indicated by thick circles), corresponding to the maximal perfect haplotype block 0101.

Twin vertices of a graph G can be determined by constructing its modular decomposition tree, where internal nodes labeled as *series* or *parallel* correspond to last descendant leaves which are twin vertices in G. McConnell and Montgolfier [7] proposed an algorithm to build a modular decomposition tree of a graph with $|V|$ vertices and $|E|$ edges that runs in $O(|V|+|E|)$ time. Since there are $O(n^2)$ necessary subgraphs to detect twin vertices, so far, such strategy is not better than the one we proposed in Algorithm 1. However, it might be possible to achieve some improvement using the fact that intervals in W are not independent.

Another alternative approach could be to use a generalized suffix tree of all the input sequences or the positional Burrows Wheeler Transform [5, 8].

References

1. 1000 Genomes Project Consortium, Auton, A., Brooks, L.D., Durbin, R.M., Garrison, E.P., Kang, H.M., Korbel, J.O., Marchini, J.L., McCarthy, S., McVean, G.A., Abecasis, G.R.: A global reference for human genetic variation. Nature **526**(7571), 68–74 (2015)
2. Browning, S.R., Browning, B.L.: Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. The American Journal of Human Genetics **81**(5), 1084–1097 (2007)
3. Chen, H., Hey, J., Slatkin, M.: A hidden Markov model for investigating recent positive selection through haplotype structure. Theoretical Population Biology **99**, 18–30 (2015)
4. Chen, H., Slatkin, M.: Inferring selection intensity and allele age from multilocus haplotype structure. G3: Genes, Genones, Genetics **3**(8), 1429–1442 (2013)
5. Durbin, R.M.: Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). Bioinformatics **30**(9), 1266–1272 (2014)
6. Hayden, E.C.: Technology: The $1,000 genome. Nature **507**(7492), 294–295 (2014)
7. McConnell, R.M., De Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Applied Mathematics **145**(2), 198–209 (2005)
8. Norri, T., Cazaux, B., Kosolobov, D., Mäkinen, V.: Minimum Segmentation for Pan-genomic Founder Reconstruction in Linear Time. In: Proceedings of WABI 2018. LIPIcs, vol. 113, pp. 15:1–15:15 (2018)
9. O'Leary, N.A., Wright, M.W., Brister, J.R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., Farrell, C.M., Goldfarb, T., Gupta, T., Haft, D., Hatcher, E., Hlavina, W., Joardar, V.S., Kodali, V.K., Li, W., Maglott, D., Masterson, P., McGarvey, K.M., Murphy, M.R., O'Neill, K., Pujar, S., Rangwala, S.H., Rausch, D., Riddick, L.D., Schoch, C., Shkeda, A., Storz, S.S., Sun, H., Thibaud-Nissen, F., Tolstoy, I., Tully, R.E., Vatsan, A.R., Wallin, C., Webb, D., Wu, W., Landrum, M.J., Kimchi, A., Tatusova, T., Dicuccio, M., Kitts, P., Murphy, T.D., Pruitt, K.D.: Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. Nucleic Acids Research **44**(D1), D733–45 (2016)
10. Pedersen, B.S., Quinlan, A.R.: cyvcf2: fast, flexible variant analysis with Python. Bioinformatics **33**(12), 1867–1869 (2017)
11. Ségurel, L., Bon, C.: On the Evolution of Lactase Persistence in Humans. Annual Review of Genomics and Human Genetics **18**, 297–319 (2017)