Minimum Recombination Histories by Branch and Bound

Rune B. Lyngsø[1], Yun S. Song[2], and Jotun Hein[1]

[1] Dept. of Statistics, Oxford University,
Oxford, OX1 3TG, United Kingdom
{lyngsoe, hein}@stats.ox.ac.uk

[2] Dept. of Computer Science, University of California, Davis, CA 95616, U.S.A.
yssong@cs.ucdavis.edu

Abstract. Recombination plays an important role in creating genetic diversity within species, and inferring past recombination events is central to many problems in genetics. Given a set M of sampled sequences, finding an evolutionary history for M with the minimum number of recombination events is a computationally very challenging problem. In this paper, we present a novel branch and bound algorithm for tackling that problem. Our method is shown to be far more efficient than the only preexisting exact method, described in [1]. Our software implementing the algorithm discussed in this paper is publicly available.

1 Introduction

Recombination is a fundamental biological process that plays a central role in generating genetic diversity within species. A question that has been receiving considerable interest, from biologists and mathematical scientists alike, is determining the minimum number $R_{\min}(M)$ of recombination events needed in the evolution of a given set M of sequences. A closely-related problem of tantamount interest is reconstructing a possible evolutionary history for M with exactly that many recombination events. Finding $R_{\min}(M)$ for an arbitrary data set M is a very difficult problem, however, and therefore several methods of computing lower bounds on $R_{\min}(M)$ have been proposed [2,3,4,5,6,7]. Such deterministic methods have interesting practical applications. For example, it has been shown [8] that efficient lower bound methods can be quite useful for detecting potential recombination hotspots. A challenging algorithmic problem is to develop new efficient methods that can produce good estimates of $R_{\min}(M)$, while being able to handle large data sets at genomic scale. For making significant progress in that direction, it would be of great help if we could study systematically when currently existing methods produce poor bounds on $R_{\min}(M)$.

A point pertinent to all lower bound methods is that, in most cases, it is difficult to know whether the number one obtains is actually equal to $R_{\min}(M)$, or whether one should try harder to obtain a sharper lower bound. To address this problem, a method of computing upper bounds on $R_{\min}(M)$ has recently been proposed [7], the main motivation being that if the upper bound is equal to the lower bound, then one actually knows the minimum number of recombination events for a given data set. Lower and upper bounds constructed in [7] are surprisingly often very close or equal when

R. Casadio and G. Myers (Eds.): WABI 2005, LNBI 3692, pp. 239–250, 2005.

recombination and mutation rates are low, but they begin to diverge as those parameters are increased, and it becomes difficult to know where in-between the two bounds $R_{\min}(M)$ lies.

Unfortunately, no efficient algorithm for computing $R_{\min}(M)$ is currently known. The only work so far that has tried to compute $R_{\min}(M)$ exactly is [1]. Being computationally intensive, both in terms of time and space, that method can analyse at most 9 sequences after some data reduction, and therefore does not have a wide range of application in practise. In this paper, we propose a branch-and-bound-based method of computing $R_{\min}(M)$ exactly that is far more efficient than the method described in [1]; our method can handle tens of sequences, requires far less memory, and runs thousands of times faster. The root sequence can be chosen to be either known or unknown in our method. In addition to finding the minimum number $R_{\min}(M)$, our method explicitly constructs a minimal ancestral recombination graph [9] that represents a possible evolutionary history with exactly $R_{\min}(M)$ recombination events; that most parsimonious (or minimal) history can be viewed using open source graphics interfaces. Our method has been fully implemented in the programming language C. The software is called `beagle` and is publicly available.

Our work should have a number of important applications. For example, using simulated data generated under various mutation and recombination rates, one can use our method to find out, when lower and upper bounds on $R_{\min}(M)$ do not match, which bound is closer to the minimum number. Such study should prove useful for devising new efficient recombination detection methods that can produce qualitatively better estimates of $R_{\min}(M)$. The minimal ARG constructed by our method should also be of interest to many researchers who wish to see an explicit graphical representation of the most parsimonious history. For instance, the upper bound method proposed in [7] explicitly constructs a class of ARGs, which may not contain minimal ARGs. By studying the minimal ARG constructed by our method when the upper bound is not equal to $R_{\min}(M)$, one may be able to learn how to capture the structure of minimal ARGs more accurately.

The organisation of this paper is as follows. In Sect. 2 we introduce the model assumed for the inference of $R_{\min}(M)$. In Sect. 3 we present our method for computing $R_{\min}(M)$. In Sect. 4 we apply our method to the data set analysed in [1] and compare performances. Finally, in Sect. 5 we discuss some future directions and open problems.

2 The Infinite Sites Model

We assume the standard infinite sites model of mutation [10]. This model is applicable to phased single nucleotide polymorphism (SNP) data sampled from a population with a relatively low mutation rate. The infinite sites model restricts the occurrence of mutation event to at most one per site. Hence, each polymorphism, or *segregating site*, is caused by exactly one mutation event, and sampled sequences can be represented as binary sequences, as each segregating site contains only two of the four possible nucleotides.

In our work, we allow sequences to contain unresolved positions. Some SNP data sets do contain sequences with unresolved positions (i.e. missing data). More importantly, as described in Sect. 3.2, recombination introduces uncertainty regarding the state of ancestral sequences. Hence, we assume that data sets consist of m sequences of length n over $\{0, 1, *\}$ (or equivalently an $m \times n$ binary matrix with entries possibly left unspecified). A "$*$" at a site corresponds to an unresolved character, while 0 and 1 correspond to the two observed nucleotides. If the grand most recent common ancestor is known, we adopt the convention that, at each site, 0 corresponds to the ancestral type while 1 corresponds to the mutant type, i.e., the grand most recent common ancestor is the all-0 sequence.

In what follows, we use M to denote the data set being analysed, with m denoting the number of sequences and n the number of segregating sites. We assume that the observed sequences have evolved from a single common ancestor through a succession of three types of events.

- a mutation event at a site that has not been subjected to mutation before, changing the state of the site in a sequence from 0 to 1 (or possibly from 1 to 0, if the most recent common ancestor is not known).
- a coalescent event, creating an extra copy of a sequence
- a recombination event, replacing two sequences with a recombinant consisting of a prefix from one sequence concatenated with the corresponding suffix of the other sequence.

As exactly one mutation event occurs in each segregating site, any valid history for M contains exactly n mutation events. Each coalescent event increases the number of sequences by one, while each recombination event decreases the number of sequences by one. Obtaining m sequences from a single common ancestor therefore require $m-1+r$ coalescent events if r recombination events occur. So the histories with the minimum number of events are the ones with the minimum number of recombination events. We denote the minimum number of recombinations required to explain a data set M by $R_{\min}(M)$.

3 Finding a Minimum Recombination History

Given a data set M, it is relatively straightforward to determine whether it can be explained by an evolutionary history with no recombinations, and to reconstruct such a history [11]. Two sites are said to be *conflicting* if they contain all four possible gametic types 00, 01, 10, and 11 (or the three types 01, 10, and 11, if the ancestral sequence is known). At least one recombination event must have occurred between two conflicting sites. Testing for the presence of the above gametic types is known as the four (or three, if the ancestral sequence is known) gamete test. Based on this test, [2] proposes a method for finding a lower bound on the number of recombinations required: a set of pairs of conflicting sites spanning non-overlapping regions implies a lower bound on $R_{\min}(M)$ equal to the set size, as at least one recombination is required between each pair of conflicting sites. In [2] it is shown how to find such a set of maximum size. However, determining the exact value of $R_{\min}(M)$ for an arbitrary data set M is $\mathbf{NP}$ hard when the most recent common ancestor is known [12].

Recently, improved methods for computing lower bounds on $R_{\min}(M)$ have been proposed [3, 4, 5, 6, 7], and in certain restricted cases, efficient methods exist for finding an evolutionary history with the minimum number of recombinations [13, 7]. We are only aware of one implemented method for computing $R_{\min}(M)$ exactly [1]. This method is based on scanning M from left to right, detecting recombinations that are needed to change local tree topologies. The main result of our present paper is a branch and bound algorithm that searches for an evolutionary history with the minimum number of recombinations. Our method constructs histories backward in time starting from the input data M, until a single common ancestor is reached. For ease of exposition, we assume that the ancestral sequence is known, but everything immediately extends to the case in which the ancestral sequence is unknown.

3.1 Reimplementation of the Haplotype Bound

Crucial elements of a branch and bound algorithm are the quality of the bounds computed and the speed with which they are obtained. We have chosen to use the haplotype bound introduced in [3]. It can provide quite powerful bounds as documented in [7], tremendously improving on the lower bound method of [2]. Though $\mathbf{NP}$ hard to compute [6], heuristics can significantly reduce computation time without seriously sacrificing power. It should be mentioned that the branch and bound algorithm implementation makes no assumptions about the lower bound computation, hence it is straightforward to substitute in any other lower bound method or implementation.

The haplotype bound can be seen as a generalisation of the method in [2]. Whereas the method of [2] looks only at pairs of segregating sites to establish a lower bound on the number of recombinations required in a region, the haplotype bound looks at all subsets of sites in the region. The key observation is that each mutation event and recombination event can introduce at most one extra sequence type. The number of mutation events equals the number of sites, due to the infinite sites assumption. Hence, if a subset of a sites gives rise to b distinct sequences, at least $b - a - 1$ recombination events must have occurred in the interval spanned by the sites. Local lower bounds obtained this way are then combined to produce a global lower bound for $R_{\min}(M)$ by determining the maximum sum of lower bounds on a set of disjoint intervals (in [3] this second step is called the composite bound, while the term haplotype bound is reserved for the first step). The authors of [3] originally implemented the haplotype bound in a program called `RecMin`. In the following we discuss our reimplementation of the haplotype bound.

If M contains identical sites, some subsets of sites can be discarded a priori: in general, we want subsets to contain as few sites and span as short intervals as possible. Let c_i denote site i of M, and let $\mathcal{C} \subseteq \{1, \ldots, n\}$ be a set of sites. We can ignore $\mathcal{C}$ if there exists $i, j \in \mathcal{C}$ with $c_i = c_j$. The set $\mathcal{C} \setminus \{i\}$ (or equivalently $\mathcal{C} \setminus \{j\}$) will give rise to as many sequence types as $\mathcal{C}$ using one site less. This provides a better lower bound in any region containing $\mathcal{C}$. Let $\operatorname{span}(\mathcal{C})$ denote the interval spanned by a set of sites $\mathcal{C}$, i.e. $\operatorname{span}(\mathcal{C}) = [\min_{i \in \mathcal{C}} .. \max_{i \in \mathcal{C}}]$. Assume there exists another set of sites $\mathcal{C}'$ containing the same types of sites, i.e. $\{c_i \mid i \in \mathcal{C}\} = \{c_i \mid i \in \mathcal{C}'\}$. If $\operatorname{span}(\mathcal{C}') \subset \operatorname{span}(\mathcal{C})$ we can ignore $\mathcal{C}$ as $\mathcal{C}'$ provides an identical lower bound in any region containing $\mathcal{C}$.

It follows that it suffices to determine the minimal intervals spanned by sets of sites corresponding to a set of site types, for all sets of type sites. We do this by incrementally

Algorithm 1. Minimal intervals combining a position from s and an interval from t

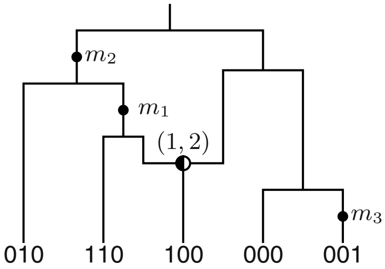
```
while $s$ and $t$ are non-empty do
  while $|s| > 1$ and $s[2] < t[1]$['right'] do
    $s = s[2:]$
  while $|t| > 1$ and $t[2]$['right'] $< s[1]$ do
    $t = t[2:]$
  if $s[1] < t[1]$['left'] then
    Report minimal interval {'left' : $s[1]$, 'right' : $t[1]$['right']}
    $s = s[2:]$
  else
    if $t[1]$['right'] $< s[1]$ then
      Report minimal interval {'left' : $t[1]$['left'], 'right' : $s[1]$}
    else
      Report minimal interval $t[1]$
    $t = t[2:]$
```

```
0000
0010
0001
1000
0100
1010
0101
1234
```

Fig. 1. An example where our heuristic fails. The exact haplotype lower bound on the set is $7 - 4 - 1 = 2$ recombinations. The only conflicting pairs of sites in the data are $\{1, 3\}$ and $\{2, 4\}$. Therefore, our heuristic fails to build the full set of all four sites, and only establish lower bounds of one recombination on the three non-disjoint regions from site 1 to site 3, from site 2 to site 4, and from site 1 to site 4.

constructing sets of type sites, adding one type site at a time. Algorithm 1 shows how we maintain the ordered set of minimal intervals spanned under the addition of a new site type, in time linear in the total number of previous minimal intervals and occurrences of the new site type.

To limit computation time, the default setting of `RecMin` is to only consider subsets of up to five sites spanning regions of width at most 12. Depending on the data, this heuristic may significantly decrease the computed bound. The user is advised to start with reasonable values of these parameters, and then increase them until the computed bound appears stable or the run time becomes too large. This approach is not well suited as a subprocess of a branch and bound algorithm. Instead, we use a more adaptive heuristic. Recall that a subset of a sites giving rise to b distinct sequences results in a bound of $b - a - 1$. If adding a new site type only increases the number of distinct sequences by at most one, the bound does not improve. We use this as a stopping criteria, ceasing further expansion of a set of site types the first time an increased bound is not obtained. It significantly reduces running time but in most cases results in the same lower bound on the minimum number of recombinations as the exact haplotype bound. As shown by the example in Fig. 1, however, data sets causing this heuristic to perform arbitrarily poorly can be constructed.



1	2	3	4	5	6	7	8	9	10
010	010	010	010	010	010	010	010	000	000
110	110	110	110	110	010	*00	000	000	
100	100	100	1**	*00	*00	000			
000	000	000	*00	000	000				
001	000		000						

Fig. 2. One possible evolutionary history, or ARG, explaining the small data set $\{010, 110, 100, 000, 001\}$. Mutations are represented by •s, where m_i denotes a mutation at the i'th site; coalescent events are represented by two lineages merging; recombination events are represented by one lineage forking out into two lineages at a ◐, where $(i, i+1)$ denotes that the crossover point is between site i and site $i+1$ and the orientation of the black half of the ◐ indicates the lineage supplying the prefix to the recombinant sequence. Listed in the table are the 10 ancestral states encountered in this ARG, with the most recent first.

3.2 Basic Branch and Bound Algorithm

We trace the evolution of M backward in time. The configurations encountered are called *ancestral states* and the aim is to reach the ancestral state consisting of only the all-0 common ancestor sequence. For a configuration ψ, the possible predecessors ψ' can be deduced from the event types listed in Sect. 2:

Mutation can occur in site i if exactly one sequence $s \in \psi$ carries the mutant type in site i; i.e. if $s[i] = 1$ and $\forall t \in \psi \setminus \{s\} : t[i] \neq 1$, then $\psi' = \psi \setminus \{s\} \cup \{s[1..i-1]0s[i+1..n]$ is a possible predecessor of ψ.

Coalescence can occur if $s, t \in \psi$ are *compatible*; i.e. if $\forall 1 \leq i \leq n : \{s[i], t[i]\} \neq \{0, 1\}$, then $\psi' = \psi \setminus \{s, t\} \cup \{u\}$ is a possible predecessor of ψ, where $u[i] = s[i]$ if $s[i] \neq *$ and $u[i] = t[i]$ otherwise.

Recombination can always occur in any sequence; if $s \in \psi$ and $0 \leq i \leq n$, then $\psi' = \psi \setminus \{s\} \cup \{s[1..i]*^{n-i}, *^i s[i+1..n]\}$ is a possible predecessor of ψ. Note that we only trace the ancestral material observed in M, so sites not passing ancestral material on to ψ are left unresolved indicated by the $*$ character. Moreover, it is easy to see that a recombination introducing the all-$*$ sequence cannot be part of an evolutionary history with a minimum number of recombinations. Hence, we implicitly assume that only recombinations splitting the recombinant into two sequences both carrying ancestral material are considered.

A sequence of events, or evolutionary history, can graphically be represented by an *ancestral recombination graph* (ARG) as illustrated in Fig. 2.

Based on this description of the consequences of each type of evolutionary event, it is easy to search for an evolutionary history with at most r recombinations. To avoid repeating previous work, we maintain a hash table containing the best lower bound established for all ancestral states already encountered. So when we arrive at an ancestral state ψ with r' recombinations remaining to meet the overall target of r recombinations we

- first check whether ψ is already present in the hash table with a lower bound exceeding r'; if this is the case we backtrack the search
- if ψ is not present in the hash table, we compute a lower bound on the minimum number of recombinations required for ψ as described in Sect. 3.1; if this lower bound exceeds r' we insert ψ in the hash table with this lower bound and backtrack the search
- otherwise, we make a depth-first search starting at the possible predecessors of ψ, with r' recombinations remaining for predecessors obtained through coalescence and mutation events and $r' - 1$ recombinations remaining for predecessors obtained through recombination events; if this search fails, we update the hash table to contain ψ with lower bound $r' + 1$.

To determine the minimum number of recombinations required for a data set M, we start by computing a lower bound r on this number as described in Sect. 3.1. We then try to find an evolutionary history with at most r recombinations. If this fails, we increase r by one. We repeat this until r is increased to $R_{\min}(M)$, at which stage we will find a minimum recombination history.

This approach essentially develops the true minimum number of recombinations required by improving a lower bound. An alternative would be to start with an upper bound r on the number of recombinations required – obtained by heuristic means, see e.g. [7], or possibly even ∞ – and updating it whenever a new evolutionary history with a lower number of recombinations is discovered. This continues until it is established that no evolutionary history with fewer recombinations than the current value of r is possible. This alternative approach would essentially develop the true minimum number of recombinations required by successive improvements of an upper bound. We would expect these two alternatives to have comparable run times, although our approach would probably exhibit a smaller variance; the total time will usually be dominated by the time used to establish that no evolutionary history exists with $R_{\min}(M)-1$ recombinations.

3.3 Refinements of the Branch and Bound Algorithm

In the previous section we described a vanilla flavoured branch & bound algorithm for determining $R_{\min}(M)$, utilising the haplotype bound. This forms the basis of our full branch & bound algorithm, but would by itself not be sufficiently efficient to yield a major improvement in efficiency compared to [1]. Hence, we apply three tricks to speed up the computation and decrease the amount of memory used: one based on data set reduction, one based on only considering certain orderings of events, and one based on requiring splits caused by recombinations to be maximal.

Reduced Data Sets. As observed in [3, 1], an ancestral state ψ can be *reduced* without changing the value of $R_{\min}(\psi)$. We use l to refer to the sequence length in ψ, which may differ from n as the reduction process may remove sites. We reduce ψ by repeatedly applying the following operations until no operations are possible:

1. collapse identical sequences into one; i.e. if $s, t \in \psi$ and $s = t$, then ψ can be replaced by $\psi' = \psi \setminus \{s\}$

```
00100000010000010010101110111101010101000000
00000000100000010011101110111101010101000000
00100000100000010000000000000001010111000101
00100000100000011100000000000001010111011000
00111000001100101100000000000001010100000000
00000000100000000000000000000000000010000010
00000000100000000000000000000101010000000000
11011000001110000000000000000000101010001000
11011000001110000000000000000000101010001000
11011000001110000000000000000000101010001000
11011111000001000000010001000010101000000000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
```

```
0100010101010101
0001010101010101
0101010001010111
0101011001010111
0110101001010100
0001000000000010
0001000010101000
1010100010101000
1010000010101000
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Fig. 3. The *Drosophila melanogaster* alcohol dehydrogenase data from [14], and the same data after having been reduced

2. remove sites where at most one sequence carries the mutant type, also known as *uninformative* sites; i.e. if $\exists 1 \leq i \leq l : |\{s \in \psi \mid s[i] = 1\}| = 1$, then ψ can be replaced by $\psi' = \{s[1..i-1]s[i+1..l] \mid s \in \psi\}$
3. collapse neighbouring identical sites; i.e. if $\exists 1 \leq i < l, \forall s \in \psi : s[i] = s[i+1]$, then ψ can be replaced by $\psi' = \{s[1..i-1]s[i+1..l] \mid s \in \psi\}$

In the presence of unresolved sites we can slightly strengthen operation 1. We will say that a sequence s is *subsumed* in another sequence t if $s[i] \in \{*, t[i]\}$ for all $1 \leq i \leq l$. That is, if s and t are compatible and s does not carry ancestral material in any site where t does not carry ancestral material. Operation 1 can be applied to any pair of sequences where one is subsumed in the other. We can define a site being subsumed in another site in a similar manner, and strengthen operation 3 to apply to neighbouring sites where one is subsumed in the other. That the effect of this reduction can be significant is illustrated in Fig. 3.

Restricted Event Orderings. Reduction can be seen as imposing restrictions on the order in which events occur. Operations 1 and 2 can be seen as equivalent to only considering evolutionary histories where coalescent events, that as a net result removes all traces of one of the sequences, and mutations are carried out at the first given opportunity. Operation 3 can be seen as requiring the mutations in the collapsed sites to happen consecutively. We can impose further restrictions on the sequences of events considered, while guaranteeing that we will still consider at least one sequence of events containing a minimum number of recombinations.

As operation 2 of ancestral state reduction forces all mutations possible, we never perform a recombination event on an ancestral state where a mutation is possible. It is easy to see that a recombination event cannot make a new mutation event possible. So after a sequence of recombination events starting from ancestral state ψ, a coalescence event must necessarily follow. This event will coalesce sequences $t = *^i s[i+1..j]*^{l-j}$ and $t' = *^{i'} s'[i'+1..j']*^{l-j'}$ (with l denoting sequence length in ψ), i.e. segments of ancestral material carried by sequences $s, t \in \psi$. Note that i, i' may equal 0 and j, j' may equal l. Evidently, any recombination not involved in creating the segments of t and t' can be postponed until after coalescing t and t'.

Even a recombination involved in creating t or t' can be postponed. Only the at most two recombinations defining the boundaries of overlap of ancestral material between t

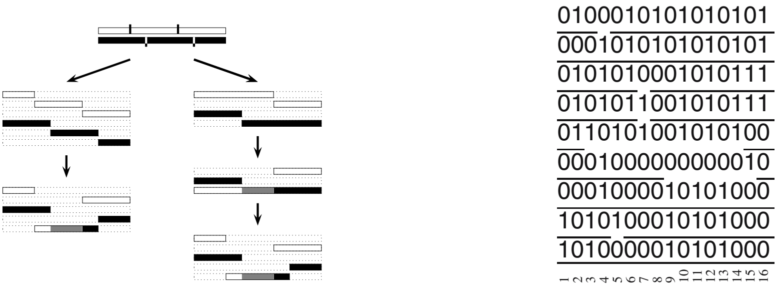


Fig. 4. Illustration of the event ordering and maximality principles used in the refined branch & bound algorithm. The right-hand illustration shows that of the four splits defining two segments being coalesced, two can be postponed till after the coalescence as illustrated in the right-hand sequence of events. In the right-hand illustration, the maximum subsumed prefix and suffix is underlined for each sequence.

and t' are required to make the coalescence possible, as illustrated in Fig. 4. Hence, we will only consider sequences of events with at most two consecutive recombination events. Moreover, the recombination event or events have to be involved in creating the sequences of the ensuing coalescence event.

Maximality. Finally, we require recombinations to split off segments that are in a certain sense maximal. We discuss this for the case where a single recombination event is followed by a coalescence involving the prefix split off by the recombination event, but the principle is applied to all recombination events. Assume the recombination splits sequence $s \in \psi$ after site i, and that the prefix $s[1..i]*^{l-i}$ (where l is the sequence length in ψ) coalesces with sequence $t \in \psi$ to create sequence t'. The ensuing ancestral state is $\psi' = \psi \cup \{*^i s[i+1..l], t'\} \setminus \{s, t\}$. If $s[i+1] = t[i+1]$ then $s[1..i+1]*^{l-i-1}$ will also coalesce with t to create t'. So by splitting s after site $i+1$ instead, we obtain the alternative ancestral state $\psi'' = \psi \cup \{*^i s[i+2..l], t'\} \setminus \{s, t\}$. Evidently, $R_{\min}(\psi'') \leq R_{\min}(\psi')$ as $*^i s[i+2..l]$ does not contain any ancestral material not present in $*^i s[i+1..l]$. Hence, we only consider recombinations that split off segments that cannot be extended without changing the outcome of the ensuing coalescence event.

More generally, for two ancestral states ϕ and ϕ' we will say that $\phi \prec \phi'$ if we can obtain ϕ from ϕ' by changing one or more characters in ϕ' to $*$. We do not pursue events leading to an ancestral state ϕ whenever it can be established that we can reach another ancestral state $\phi' \prec \phi$ using the same number of recombinations. In particular, let the *maximum subsumed prefix* of a sequence s in ancestral state ψ be the longest prefix of s that is subsumed in some other sequence of ψ. Similarly define the *maximum subsumed suffix*. Then no recombination events occurring within the maximum subsumed prefix or suffix of a sequence are considered (with the exception of one recombination event if they overlap). This is illustrated in Fig. 4.

4 Application

We have implemented the branch & bound algorithm outlined in the previous section in a C program called `beagle` (as branch & bound shares its acronym with the infamous Beagle Boys). The source code is available under the GNU public license from `http://www.stats.ox.ac.uk/~lyngsoe/beagle/`. The program can produce an evolutionary history that established the value of $R_{\min}(M)$, and accompanying scripts can convert such a history to three different formats used by different programs for drawing networks. The program can also compute the minimum number $R_{\min}(I)$ of recombinations required in a sub-interval I of the input data M. Note that these local values $R_{\min}(I)$ may not be attainable by any minimal ARG for the entire data M.

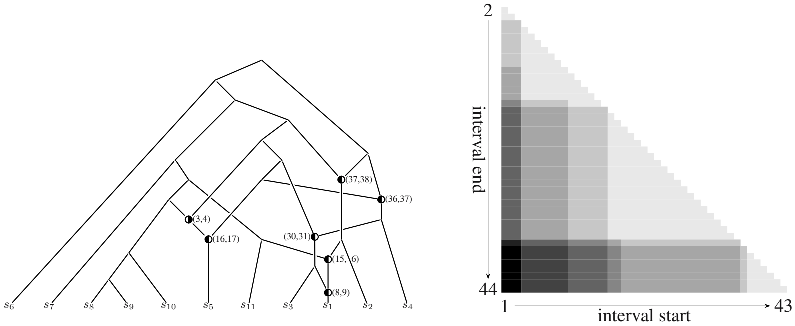


Fig. 5. Results obtained from applying `beagle` to the data of [14]. On the left is the ARG representing the evolutionary history with $R_{\min}(M) = 7$ found by `beagle`, with mutations left out for clarity. On the right is the matrix of local values of $R_{\min}(I)$ for all sub-intervals I of the data, colour coded from light grey for no recombinations to black for 7 recombinations.

We have applied `beagle` to Kreitman's data of the alcohol dehydrogenase locus from 11 chromosomes of *Drosophila melanogaster*. This data set was also analysed in [1], and therefore it provides a useful benchmark for comparing our method with the only previously existing method for computing $R_{\min}(M)$ exactly. The data is shown in Fig. 3. An ARG recreated from the history and the matrix of local values of $R_{\min}(I)$ are shown in Fig. 5. The implementation of the method described in [1] required about 30 minutes on a 1.26GHz Pentium III processor and 1.5GB of memory to determine that $R_{\min}(M) = 7$ for this data set. On a similar processor, a 1.4GHz Athlon processor, `beagle` obtained the same result in less than a second using less than 100kB of memory. On this particular data set, we have thus managed an improvement of three to four orders of magnitude in both time and space requirements. This significantly expands the range of data for which an exact computation of $R_{\min}(M)$ is feasible.

To illustrate how the performance of `beagle` depends on the size and complexity of the input data, we have also applied `beagle` to the human LPL data from [15], with missing or non-SNP sites removed. These sequences are sampled from three geographically distinct regions, Jackson, North Karelia, and Rochester. Following [3] we further partition the sites into three regions. Analysing the full data set is beyond the capabilities of `beagle`, but based on the above partitions we obtain nine smaller data sets of varying size and complexity. For eight of the nine data sets `beagle` was able to determine $R_{\min}(M)$ and the results are summarised in Table 1. Not surprisingly, a key aspect influencing the time requirement seems to be how far the initial lower bound

Table 1. Results of applying `beagle` to the human LPL data. In each entry, we first list the size of the data set in terms of number of sequences/number of sites; we then list $R_{\min}(M)$ with the initial lower bound obtained from our haplotype heuristic in parentheses; finally, we list the time required to determine $R_{\min}(M)$ on a 1.4GHz Athlon processor.

Population	region 1	region 2	region 3
Jackson	40/14; 13(10); 18m52s	40/14; 10(8); 26s	40/22; 15(12); $>$ 15h
North Karelia	31/15; 2(2); $<$ 1s		31/24; 8(7); $<$ 1s
Rochester	27/17; 1(1); $<$ 1s	27/13; 14(12); 8m00s	27/28; 8(7); $<$ 1s

is from $R_{\min}(M)$. Only two of these data sets are sufficiently small that it would be feasible to use the method of [1] to determine $R_{\min}(M)$.

5 Discussion

The framework of our branch & bound algorithm works independently of the lower bound method utilised. It should be quite easy to replace our implementation of the haplotype bound with other lower bound implementations available, to explore whether any of these can further improve the efficiency of `beagle`. Two crucial aspects are the quality of the bound produced and the time taken to produce it. A qualitatively improved lower bound will in general result in `beagle` having to explore fewer ancestral states, while a faster lower bound method will result in `beagle` having to spend less time on each ancestral state encountered. Profiling data obtained from running `beagle` indicates that most of CPU time is spent on computing lower bounds. A lower bound of similar quality that can be computed faster can lead to a significant reduction in computation time, but not a reduction by orders of magnitude. To obtain such a reduction we would need a method producing better lower bounds. Therefore, we are particularly interested in testing `beagle` with the improved haplotype bound discussed in [7].

It would be interesting and worthwhile to use `beagle` to produce a lower bound on $R_{\min}(M)$. We can use `beagle` to compute the minimum number of recombinations for all small regions of the data, while using an efficient lower bound method for large regions. These local bounds can then be combined using the composite bound method of [3] to produce a lower bound on $R_{\min}(M)$ for the entire data. We expect `beagle` to work quite well on data sets of short sequences, for which the search space reduction based on ignoring recombinations in maximum subsumed prefixes and suffixes should dramatically limit the number of recombination events pursued. In this context, it should be noted that the method of [1] may be more efficient for data sets consisting of very few long sequences, as the running time of that method depends only linearly on sequence length, albeit super-exponentially on the number of sequences.

The computation of a local value $R_{\min}(I)$ by `beagle` is oblivious to the region surrounding I, i.e. the local value may not be attainable in any history with a globally minimum number $R_{\min}(M)$ of recombinations. It would be of interest to compute local values in the context of the surrounding data, not least to study the effects of ignoring context. It is not clear to us, however, whether `beagle` can be modified to compute local $R_{\min}(I)$ values without sacrificing the refinements discussed in Sect. 3.3

Acknowledgements

YS is supported by grant EIA-0220154 from the National Science Foundation. JH and RL are supported by EPSRC grant HAMJW, and MRC grant HAMKA.

References

1. Song, Y.S., Hein, J.: Parsimonious reconstruction of sequence evolution and haplotype blocks. In: Proceedings of the 3rd Workshop on Algorithms in Bioinformatics (WABI). (2003) 287–302
2. Hudson, R.R., Kaplan, N.L.: Statistical properties of the number of recombination events in the history of a sample of DNA sequences. Genetics **111** (1985) 147–164
3. Myers, S.R., Griffiths, R.C.: Bounds on the minimum number of recombination events in a sample history. Genetics **163** (2003) 375–394
4. Gusfield, D., Hickerson, D.: A new lower bound on the number of needed recombination nodes in both unrooted and rooted phylogenetic networks. Technical Report UCD-ECS-06, University of California, Davis (2004)
5. Song, Y.S., Hein, J.: On the minimum number of recombination events in the evolutionary history of DNA sequences. Journal of Mathematical Biology **48** (2004) 160–186
6. Bafna, V., Bansal, V.: Improved recombination lower bounds for haplotype data. In: Proceedings of the 9th Annual International Conference on Computational Molecular Biology (RECOMB). (2005)
7. Song, Y.S., Wu, Y., Gusfield, D.: Efficient computation of close lower and upper bounds on the minimum number of recombinations in biological sequence evolution. In: Proceedings of the 13th International Conference on Intelligent Systems for Molecular Biology (ISMB). (2005) in press.
8. Fearnhead, P., Harding, R.M., Schneider, J.A., Myers, S., Donnelly, P.: Application of coalescent methods to reveal fine-scale rate variation and recombination hotspots. Genetics **167** (2004) 2067–2081
9. Griffiths, R.C., Marjoram, P.: An ancestral recombination graph. In: Progress in Population Genetics and Human Evolution. Volume 87 of IMA Volumes in Mathematics and its Applications. Springer Verlag (1997) 257–270
10. Kimura, M.: The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. Genetics **61** (1969) 893–903
11. Gusfield, D.: Efficient algorithms for inferring evolutionary trees. Networks **21** (1991) 19–28
12. Wang, L., Zhang, K., Zhang, L.: Perfect phylogenetic networks with recombination. Journal of Computational Biology **8** (2001) 69–78
13. Gusfield, D., Eddhu, S., Langley, C.: Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. Journal of Bioinformatics and Computational Biology **2** (2004) 173–213
14. Kreitman, M.: Nucleotide polymorphism at the alcohol dehydrogenase locus of Drosophila melanogaster. Nature **304** (1983) 412–417
15. Nickerson, D.A., Taylor, S.L., Weiss, K.M., Clark, A.G., Hutchinson, R.G., Stengard, J., Salomaa, V., Vartiainen, E., Boerwinkle, E., Sing, C.F.: DNA sequence diversity in a 9.7-kb region of the human lipoprotein lipase gene. Nature Genetics **19** (1998) 216–7