Chapter 3
Genetic Map Construction

Gene loci are grouped into different chromosomes. Within the same chromosome, the loci are linearly arranged because the chromosome is a string-like structure. The distribution of loci among chromosomes and the order of loci within chromosomes are called the genetic map. The data used to construct the genetic map are the genotypes of these loci. From the genotypic data, we can estimate all pairwise recombination fractions. These estimated recombination fractions are used to construct the linkage map. Construction of the genetic map may be better called reconstruction of genetic map because the true genetic map is already present and we simply do not know about it. This is similar to the situation where phylogeny construction is more often called phylogeny reconstruction because we are not constructing the phylogeny of species; rather, we infer the existing phylogeny using observed data. Of course, the inferred map may not be the true one if the sample size is not sufficiently large. Map construction is the first step toward gene mapping (locating functional genes). There are two steps in genetic map construction. The first step is to classify markers into linkage groups according to the pairwise LOD scores or the likelihood ratio test statistics. A convenient rule is that all markers with pairwise LOD scores greater than 3 are classified into the same linkage group. A more efficient grouping rule may be chosen using a combination of LOD score and the recombination fraction. For example, loci A and B may be grouped together if $\mathtt{LOD}_{AB} > 3$ and $r_{AB} < 0.45$. Grouping markers into the same linkage group is straightforward, and no additional technique is required other than comparing the LOD score of each pair of markers to a predetermined LOD criterion. If we choose a more stringent criterion, some markers may not be assigned into any linkage groups. These markers are called satellite markers. On the other hand, if we choose a less stringent criterion, markers on different chromosomes may be assigned into the same linkage group. The second step of genetic mapping is to find the optimal orders of the markers within the same linkage groups. In this chapter, we only discuss the second step of map construction.

S. Xu, *Principles of Statistical Genomics*, DOI 10.1007/978-0-387-70807-2_3,

3.1 Criteria of Optimality

Given the estimated pairwise recombination fractions for m loci on the same linkage group, we want to find the optimal order of the loci. There are $m!/2$ possible ways to arrange the m loci. The factorial of m gives the total number of permutations of all m loci. However, the orientation of a linkage map is irrelevant. For instance, ABC and CBA are considered the same order for loci A, B, and C, as far as the relative positions of the loci are concerned.

We will first define a criterion of "optimality" and then select the particular order that minimizes or maximizes the criterion. The simplest and also the most commonly used criterion is the sum of adjacent recombination coefficients (sar). The criterion is defined as

$$\text{sar} = \sum_{i=1}^{m-1} \hat{r}_{i(i+1)} \tag{3.1}$$

where $\hat{r}_{i(i+1)}$ is the recombination fraction between loci i and $i+1$ for $i = 1, \ldots, m-1$, where i and $i+1$ are two adjacent loci. For m loci, there are $m-1$ adjacent recombination fractions. If there is no estimation error for each of the adjacent recombination fraction, the true sar should have the minimum value compared with any other orders. Consider the following example of three loci with the correct order of ABC. The sar value for this correct order is

$$\text{sar}_{ABC} = r_{AB} + r_{BC} \tag{3.2}$$

If we evaluate an alternative order, say ACB, we found that

$$\text{sar}_{ACB} = r_{AC} + r_{BC} \tag{3.3}$$

Remember that the true order is ABC so that $r_{AC} = r_{AB} + r_{BC} - 2r_{AB}r_{BC}$ assuming that there is no interference. Substituting r_{AC} into the above equation, we get

$$\begin{aligned} \text{sar}_{ACB} &= r_{AB} + r_{BC} - 2r_{AB}r_{BC} + r_{BC} \\ &= r_{AB} + r_{BC} + r_{BC}(1 - 2r_{AB}) \end{aligned} \tag{3.4}$$

Because $r_{BC}(1 - 2r_{AB}) \geq 0$, we conclude that $\text{sar}_{ACB} \geq \text{sar}_{ABC}$. In reality, we always use estimated recombination fractions, which are subject to estimation errors, and thus the marker order with minimum sar may not be the true order.

Similar to sar, we may use sad (sum of adjacent distances) as the criterion, which is defined as

$$\text{sad} = \sum_{i=1}^{m-1} \hat{x}_{i(i+1)} \tag{3.5}$$

where $\hat{x}_{i(i+1)}$ is the estimated additive distance between loci i and $i+1$ and is converted from the estimated recombination fraction using either the Haldane or Kosambi map function. Similar to the *sar* criterion, the order of loci that minimizes *sad* is the optimal order.

The sum of adjacent likelihoods (*sal*) is another criterion for map construction. Note that the likelihood refers to the log likelihood. In contrast to sar, the optimal order should be the one which maximizes sal. Define $L(\hat{r}_{i(i+1)})$ as the log likelihood value for the recombination fraction between loci i and $i+1$. The sal is defined as

$$\text{sal} = \sum_{i=1}^{m-1} L(\hat{r}_{i(i+1)}) \tag{3.6}$$

Both sad and sal are additive, which is a property required by the branch and bound algorithm for searching the optimal order of markers (see next section).

3.2 Search Algorithms

3.2.1 Exhaustive Search

Exhaustive search is an algorithm in which all possible orders are evaluated. As a result, it guarantees to find the optimal order. Recall that for m loci, the total number of orders to be evaluated is $n = m!/2$. The number of orders (n) grows quickly as m increases, as shown in the following table.

m	n
2	1
3	3
4	12
5	60
6	360
7	2,520
8	20,160
9	181,440
10	1,814,400

The algorithm will use up the computing resource quickly as m increases. Therefore, this algorithm is rarely used when $m > 10$. When writing the computer code to evaluate the orders, we want to make sure that all possible orders are evaluated. This can be done using the following approach. Assume that there are five loci, denoted by A, B, C, D, and E, that need to be ordered. First, we arbitrarily choose two loci, say A and B, to initiate the map. We then add locus C to the existing map. There are three possible places where we can put C in the existing map: CAB, ACB, and ABC. For each of the three orders of the three-locus map, we add locus D. For example, to add locus D to the existing order ACB, we need to evaluate the following four possible orders: DACB, ADCB, ACDB, and ACBD. We then add locus E (the last locus) to each of the four orders of the four-locus map.

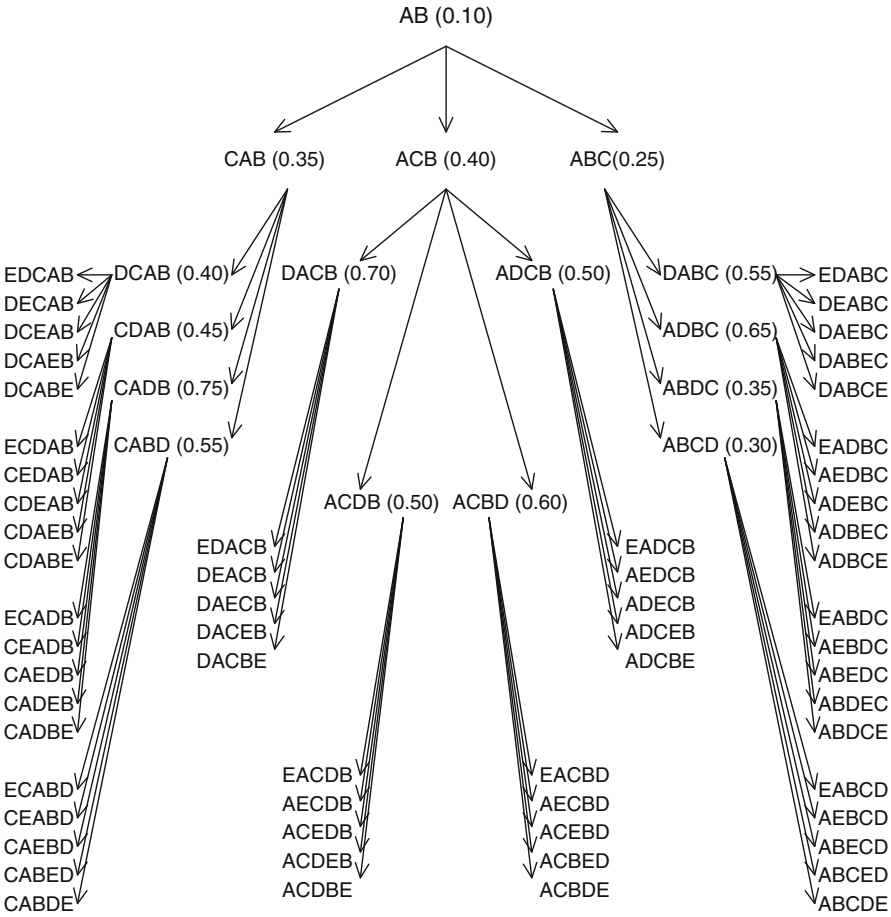

Fig. 3.1 All possible orders (60) of a map with five loci

For example, locus E can be added to the existing order DACB in five different places, leading to EDACB, DEACB, DAECB, DACEB, and DACBE. We can see that all the $3 \times 4 \times 5 = 5!/2 = 60$ possible orders have been evaluated so far (see Fig. 3.1). This ends the exhaustive search.

3.2.2 Heuristic Search

When the number of loci is too large to permit the exhaustive search, the optimal order can be sought via a heuristic approach that sacrifices the guarantee of optimality in favor of the reduced computing time. With a heuristic search, one starts with an arbitrary order and evaluates this particular order. The order is then

rearranged in a random fashion and reevaluated. If the new order is "better" than the initial order, the new order is accepted. If the new order is "worse" than the initial order, the initial order is kept. This completes one cycle of the search. The second cycle of the search starts with a rearrangement of the order of the previous cycle. The rearranged order is then accepted or rejected depending on whether or not the value of the order has been improved. The process continues until no further improvement is achieved for a certain number of consecutive cycles, e.g., 50 cycles. This method is also called the greedy method because it is too greedy to "climb up" the hill. It is likely to end up with a local optimum instead of a global one. Therefore, there is no guarantee that the method finds the global optimal order.

The so-called random rearrangement may be conducted in several different ways. One is called complete rearrangement or global rearrangement. This is done by selecting a completely different order by random permutation. Information of the previous order has no effect on the selection of the new order. This approach is conceptually simple but may not be efficient. For example, if a previous order is already close to the optimal one, a complete random rearrangement may be far worse than this order, leading to many cycles of random rearrangements before an improved order appears. The other way of rearranging the order is called partial or local rearrangement. This is done by randomly rearranging a subset of the loci. Let $m_s (2 \leq m_s < m)$ be the size of the subset. Although m_s may be chosen in a arbitrary fashion, $m_s = 3$ may be a convenient choice. First, we randomly choose a triplet from the m loci. We then rearrange the three loci within their existing positions and leave the order of the remaining loci intact. There are $3! = 6$ possible ways to rearrange the three loci, and all of them are evaluated. The best one of the six is chosen as a candidate new order for reevaluation. If this new order is better than the order in the previous cycle, the order is updated; otherwise, the previous order is carried over to the next cycle.

The result of heuristic search depends on the initial order selected to start the search. We will use the five-locus example to demonstrate a simple way to choose the initial map order. Let A, B, C, D, and E be the five loci. We start with two most closely linked loci, say loci A and D. We then add a third locus to the existing two-locus map. The third locus is chosen such that it has the minimum average recombination fractions from loci A and D. Assume that locus C satisfies this criterion. We then add locus C to the existing map AD. There are three places where locus C can be added: CAD, ACD, and ADC. Choose the best of the three orders as the optimal map, say ACD. We then choose a next locus to add to the existing map ACD, using the same criterion, i.e., minimum average recombination fractions from loci A, C, and D. Assume that locus E satisfies this criterion. There are four places that locus E can be inserted into the existing map ACD, which are EACD, AECD, ACED, and ACDE. Assume that EACD is the best of the four orders. Finally, we add locus B (the last locus) to the four-locus map. We evaluate all the five different orders: BEACD, EBACD, EABCD, EACBD, and EACDB. Assume that BEACD is the best of the five orders. This order (BEACD) can be used as the initial order to start the heuristic search.

3.2.3 Simulated Annealing

Simulated annealing is a method which examines a much larger subset of the possible orders. The method was developed to prevent the solution from being trapped into a local optimum. Like the heuristic search, we start with an arbitrary order and evaluate the order using *sar*, *sad*, or $-sal$ as the criterion. Note that *sal* is replaced by $-sal$ because the method always searches for the minimum value of the criterion. The score of the initial order is denoted by E_0. The order is then subject to local rearrangement in a random fashion. This involves Monte Carlo simulation for the order. Note that the rearrangement should be local rather than global. Assume that the score for the new order is E_1. If the new order is "better" than the initial order, i.e., $E_1 < E_0$, the new order is accepted. If the new order is "worse" than the initial order, i.e., $E_1 > E_0$, it is accepted with a probability,

$$\alpha = \exp\left(-\frac{E_1 - E_0}{k_\mathrm{b} T}\right) \tag{3.7}$$

where k_b is a physical constant called Boltsman constant and T corresponds to the temperature. Once the new order is accepted, we replace E_0 by E_1 and continue the search for another order. This sampling strategy was proposed by Metropolis et al. (1953) and thus also is referred to as the Metropolis algorithm. By trial and error, it is found that $k_\mathrm{b} = 0.95$ usually works well. However, different values should be chosen if $k_\mathrm{b} = 0.95$ does not. The value of the temperature T can be chosen arbitrarily, say $T = 2$ or any other values. For m loci, $100m$ new orders should be examined for each value of T, and then T should be changed to $k_\mathrm{b} T$ (the temperature has been lowered) at this point. With a pseudocode notation, the change of temperature is expressed as $T = k_\mathrm{b} T$ when the temperature is decided to change. The algorithm stops after 100 rearrangements have failed to provide a better order. When we write the computer code to simulate the event of accepting or rejecting a new order, we do not care about whether $E_1 < E_0$ or $E_1 > E_0$. We simply let the new order to be accepted with probability α, which is defined as

$$\alpha = \min\left[1, \exp\left(-\frac{E_1 - E_0}{k_\mathrm{b} T}\right)\right] \tag{3.8}$$

If $E_1 < E_0$, i.e., the new order is better than the old order, $\alpha = 1$, meaning that the new order is always accepted. If $E_1 > E_0$, then $\alpha < 1$; the probability of accepting the new order is not 100 %. When the temperature T gets lower, it makes the acceptance of a worse order harder. This can be shown by looking at the profile of the acceptance probability as a function of the deviation of the new order E_1 from the current order E_0 (see Fig. 3.2). The initial length of the map is $E_0 = 3$. The new length E_1 ranges from 2 to 6. The Boltsman constant is $k_\mathrm{b} = 0.95$. The three lines represent three different T values ($T = 2, 0.5, 0.1$). When $E_1 \leq E_0$, the probability of acceptance is 1. After E_1 passes $E_0 = 3$ ($E_1 > E_0$), i.e., the new order is

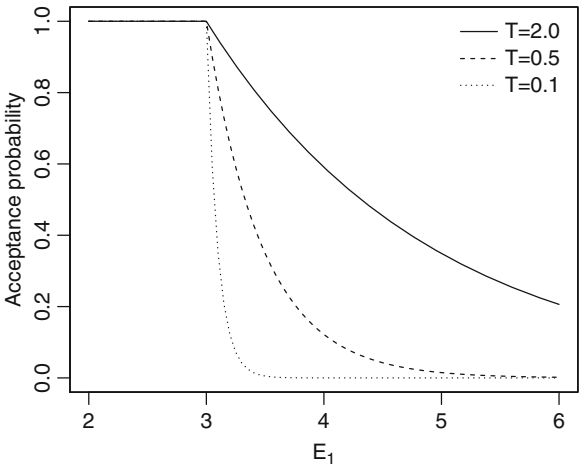

Fig. 3.2 Change of acceptance probability as E_1 deviates from $E_0 = 3.0$

worse than the current order, the acceptance probability starts to decrease but very slowly for $T = 2$ (high temperature). As the temperature cools down ($T = 0.5$), the acceptance probability decreases more sharply, meaning that it is hard to accept a worse order. When $T = 0.1$, very low temperature, the acceptance probability decreases very sharply, making the acceptance of a worse order extremely difficult. In the end, only a better order gets accepted, and no worse order will be accepted. This will end the search.

The intention of allowing a worse order to be accepted is to prevent the algorithm from settling down at a local optimal order and ignoring a global optimum elsewhere in the space of possible orders. Simulated annealing was set up in a language from the observation that when liquids are cooled very slowly, they will crystallize in a state of minimum energy (Metropolis et al. 1953).

3.2.4 Branch and Bound

The branch and bound method is often used in search for evolutionary trees (also called phylogenies). The method was first developed by Land and Doig (1960). It is adopted here to search for the optimal order of loci. This algorithm is not the exhaustive search, but it guarantees to find the global optimum order. There will be occasions when it would require examination of all orders, but generally, it requires examination of only a small subset of all possible orders. The criterion of evaluation must be "additive." This property will make sure that the map length for a particular order with k loci cannot be shortened by adding another locus to the existing map of k loci. Both *sad* and $-sal$ follow the additive rule and thus can be used as the length of a map for the branch and bound search. However, sar cannot be used here because it is not additive. Let us assume that there are four loci, ABCD, to be

ordered. Suppose that the first two loci to be considered are AB. The next locus C can be inserted in one of three positions, corresponding to orders CAB, ACB, and ABC, respectively. The fourth locus D can be inserted in four different positions for each of the three-locus orders. There will be $4!/2 = 12$ possible orders after locus D is inserted. In general, we start with two loci (one possible order) and insert the third locus to the two-locus map (three possible places to insert the third locus). When the ith locus ($i = 3, \ldots, m$) is inserted into the map of $i - 1$ loci, there will be i branch points (places) to insert the ith locus. Overall, there are $3 \times 4 \times \cdots \times m = (1 \times 2 \times 3 \times 4 \times \cdots \times m)/(1 \times 2) = m!/2$ possible orders. This process is similar to a tree growing process, as illustrated in Fig. 3.1 for the example of five loci, except that this tree is drawn upside down with the root at the top. Each tip of the tree represents a particular order of the map for all the m loci, called a child. Each branch point represents an order with $m-1$ or a lower number of loci, called a parent. The initial order of two loci is the root of the tree, called the ancestor. Each member of the tree (including the ancestor, the parents, and the children) is associated with an sad value, i.e., the length of the member.

If all the possible orders were evaluated, the method would be identical to the exhaustive search. The branch and bound method, however, starts with an arbitrarily chosen order of the m locus map (a child) and assigns the length of this order to E_0, called the upper bound. It is more efficient to select the shortest map order found from a heuristic search as the initial upper bound. Once an upper bound is assigned, it is immediately known that the optimal order cannot have a value greater than E_0. Let T_0 be the map order for the selected child (the length has been chosen as the upper bound). The branch and bound algorithm starts evaluating all the siblings of T_0. The upper bound will be replaced by the shortest length of the siblings in the family if the current T_0 is not the shortest one. Once all the siblings of the current family are evaluated, we backtrack to the parent and evaluate all the siblings of the parent (the uncles of T_0). Remember that all members in the parental generation have $m - 1$ loci. Any uncles whose scores are longer that E_0 will be disqualified for further evaluation because they will not produce children with scores shorter than E_0 due to the property that inserting additional loci cannot possibly decrease the score. Therefore, we can dispense with the evaluation of all children that descend from those disqualified uncles in the search and immediately backtrack and proceed down a different path. Only uncles whose scores are shorter than E_0 will be subject to further evaluation. The upper bound will be updated if a shorter member is found in the uncle's families. Once all the uncles and their families are evaluated, we backtrack to the great grandparent and the siblings of the grandparent and evaluate the families of all the siblings of the grandparent. The process continues until all qualified families have been evaluated. The upper bound E_0 is constantly updated to ensure that it holds the length of the shortest map order among the orders evaluated so far. Constantly updating the upper bound is important, as it may enable other search paths to be terminated more quickly.

The following example is used to demonstrate the branch and bound algorithm. Let A, B, C, and D be four loci with unknown order. The recombination fractions are stored in the upper triangular positions of the matrix given in Table 3.1. The additive

Table 3.1 Recombination fractions and additive distances for four marker loci

	A	B	C	D
A		$r_{AB}(0.0906)$	$r_{AC}(0.1967)$	$r_{AD}(0.2256)$
B	$x_{AB}(0.10)$		$r_{BC}(0.1296)$	$r_{BD}(0.1648)$
C	$x_{AC}(0.25)$	$x_{BC}(0.15)$		$r_{CD}(0.0476)$
D	$x_{AD}(0.30)$	$x_{BD}(0.20)$	$x_{CD}(0.05)$	

Table 3.2 The 12 possible orders of a four locus map and their *sad* scores

Order	Map	*sad* score
1	DCAB	0.40
2	CDAB	0.45
3	CADB	0.75
4	CABD	0.55
5	DACB	0.70
6	ADCB	0.50
7	ACDB	0.50
8	ACBD	0.60
9	DABC	0.55
10	ADBC	0.65
11	ABDC	0.35
12	ABCD	0.30

distances converted from the recombination fractions using the Haldane map function are stored in the lower triangular positions of the matrix (Table 3.1). The 12 possible orders of the loci are given in Table 3.2 along with the *sad* score for each order. For example, the *sad* score for order CABD is

$$\text{sad}_{DBAC} = x_{BD} + x_{AB} + x_{AC} = 0.20 + 0.10 + 0.25 = 0.55. \tag{3.9}$$

From Table 3.2, we can see that the optimal order is order 12, i.e., ABCD, because its *sad* score (0.30) is minimum among all other orders. This would be the result of exhaustive search because we had evaluated all the 12 possible orders. Let us pretend that we had not looked at Table 3.2 and we want to proceed with the branch and bound method to search for the optimal order.

The entire tree of four loci is given in Fig. 3.3. Each child has four loci and a length given in parentheses. Note that order ACBD has a length 0.60. The five children of ACBD do not belong to this tree of four loci. They are presented here to indicate that the tree of five loci can be expanded from the tree of four loci in this way. A randomly selected order, say DCAB, is used as T_0 whose *sad* score is used as the upper bound, $E_0 = \text{sad}_{DCAB} = 0.40$. All the siblings of DCAB are evaluated for the *sad* scores. It turns out that $\text{sad}_{DCAB} = 0.40$ is the shortest order in the family, and thus E_0 cannot be improved. The search is backtracked to CAB (the parent of DCAB), and the two siblings of the parent are evaluated, with scores of $\text{sad}_{ACB} = 0.40$ and $\text{sad}_{ABC} = 0.25$. Because $\text{sad}_{ACB} = \text{sad}_{DCAB} = E_0 = 0.40$,

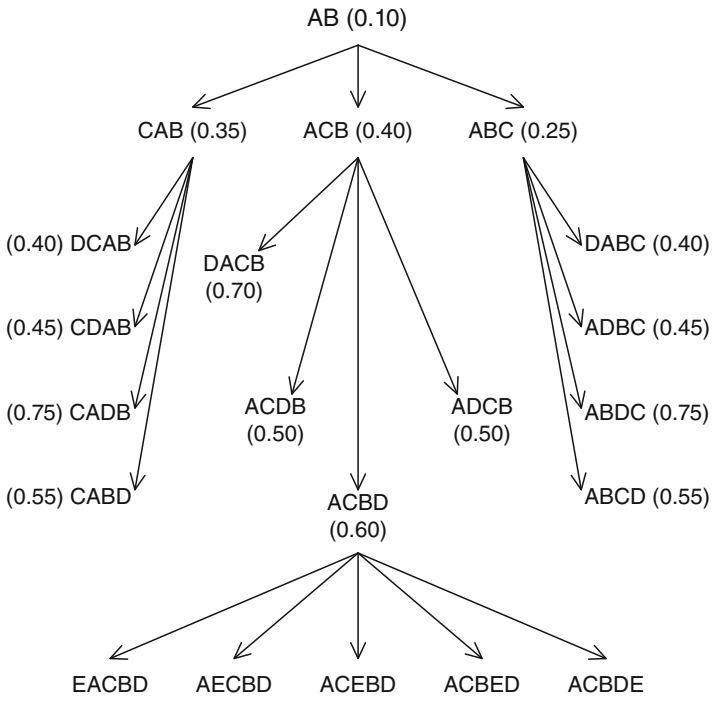

Fig. 3.3 All possible orders (12) of a map with four loci. The five-locus children of order ACBD are also given to show that the tree can be expanded this way for more loci

it is concluded that the optimal order cannot occur in the "lineage" descending from ACB because adding one more locus cannot possibly make the sad score shorter. Therefore, only children of ABC are qualified for further evaluation. The shortest order occurs in this family, and it is ABCD with $\text{sad}_{ABCD} = 0.30$. We only evaluate two families out of three $(2/3)$ to find the optimal order by using the branch and bound algorithm.

Let us use the sad of a different order as the upper bound to start the search and show that the branch and bound algorithm may end up with evaluating all possible orders. Assume that we choose the score of ACDB as the upper bound, i.e., $E_0 = \text{sad}_{ACDB} = 0.50$. We first evaluated all the siblings of ACDB and found that the upper bound cannot be improved. We then backtracked to the parent of ACDB, which is ACB. The parent has two siblings, CAB and ABC; both are shorter than E_0, and thus both should be further evaluated. However, which of the lineages is evaluated first can make a difference regarding the efficiency of the search. Assume that the lineage under CAB is evaluated first. This leads to an improved upper bound $E_0 = \text{sad}_{DCAB} = 0.40$. This upper bound is identical to the length of the parent of the family that we started the search. This family would not have been evaluated if we had chosen $E_0 = 0.40$ as the upper bound in the beginning. Unfortunately,

it was too late that we had already evaluated this family. Since ABC is shorter than $E_0 = 0.40$, the lineage under ABC is subject to further evaluation. The shortest order occurs in this lineage, which is ABCD with $\text{sad}_{ABCD} = 0.30$. All the 12 possible orders have been evaluated before the optimal order is found. However, if we had evaluated the lineage under ABC first, we would not have to evaluate the lineage under CAB, leading to a 1/3 cut of the computing load.

Finally, if the upper bound is assigned a value from a member in the ABC lineage, say $E_0 = \text{sad}_{ABDC} = 0.35$, the upper bound is immediately updated as $E_0 = \text{sad}_{ABCD} = 0.30$. This upper bound immediately disqualifies the other two lineages, leading to a 2/3 reduction of the number of orders for evaluation.

The branch and bound algorithm guarantees to find the shortest order, but the efficiency depends on the upper bound chosen and the sequence in which the paths are visited.

3.3 Bootstrap Confidence of a Map

In phylogeny analysis, an empirical confidence can be put on each internal branch of a particular phylogeny via bootstrap samplings (Felsenstein 1985). This idea can be adopted here for map construction. A map with m markers have $m - 1$ internal segments, similar to the internal branches of a phylogenetic tree. We can put a confidence on each segment. Let us assume that C–D–B–A–E is the optimal order we found. We want to put a bootstrap confidence on segment D–B. First, we draw a large number of bootstrap samples, say N. Each bootstrap sample contains n randomly sampled progeny from the original sample (map population) with replacement. This means that in a bootstrap sample, some progeny may be drawn several times while others may not be drawn at all. For each bootstrap sample, we estimate all the pairwise recombination fractions and construct a map (find the optimal order of the markers for that particular bootstrap sample). In the end, we will have N different maps, one from each bootstrap sample. We then count the number of maps that have segment D–B, i.e., D and B are joined together. The proportion of the maps that reserve this segment is the confidence of this segment. Let $N = 100$ be the number of bootstrap samples and $N_{D-B} = 95$ be the number of samples reserving segment D–B; the confidence for segment D–B is $N_{D-B}/N = 0.95$. Each segment of the map can be put a confidence using the same approach.

The way to place a bootstrap confidence for a segment of a map described here appears to be different from the bootstrap confidence of an internal branch of a phylogenetic tree. We simply adopted the idea of phylogenetic bootstrap analysis, not the way of confidence assignment. To fully adopt the bootstrap confidence assignment, we need to find the number of bootstrap samples that partition the loci into {C,D} and {B,A,E} subsets and also reserve the D–B segment. That number divided by $N = 100$ would give the confidence for the D–B segment.