4 *Exhaustive Search*

Exhaustive search algorithms require little effort to design but for many problems of interest cannot process inputs of any reasonable size within your lifetime. Despite this problem, exhaustive search, or *brute force* algorithms are often the first step in designing more efficient algorithms.

We introduce two biological problems: *DNA restriction mapping* and *regulatory motif finding*, whose brute force solutions are not practical. We further describe the *branch-and-bound* technique to transform an inefficient brute force algorithm into a practical one. In chapter 5 we will see how to improve our motif finding algorithm to arrive at an algorithm very similar to the popular CONSENSUS motif finding tool. In chapter 12 we describe two randomized algorithms, GibbsSampler and RandomProjections, that use coin-tossing to find motifs.

4.1 Restriction Mapping

Hamilton Smith discovered in 1970 that the *restriction enzyme Hin*dII cleaves DNA molecules at every occurrence, or *site*, of the sequences GTGCAC or GTTAAC, breaking a long molecule into a set of *restriction fragments*. Shortly thereafter, maps of restriction sites in DNA molecules, or *restriction maps*, became powerful research tools in molecular biology by helping to narrow the location of certain genetic markers.

If the genomic DNA sequence of an organism is known, then construction of a restriction map for *Hin*dII amounts to finding all occurrences of GTGCAC and GTTAAC in the genome. Because the first bacterial genome was sequenced twenty-five years after the discovery of restriction enzymes,

for many years biologists were forced to build restriction maps for genomes without prior knowledge of the genomes' DNA sequence.[1]

Several experimental approaches to restriction mapping exist, each with advantages and disadvantages. The distance between two individual restriction sites corresponds to the length of the restriction fragment between those two sites and can be measured by the *gel electrophoresis* technique described in chapter 2. This requires no knowledge of the DNA sequence. Biologists can vary experimental conditions to produce either a *complete* digest [fig. 4.1 (a)] or a *partial* digest [fig. 4.1 (b)] of DNA.[2] The restriction mapping problem can be formulated in terms of recovering positions of points when only pairwise distances between those points are known.

To formulate the restriction mapping problem, we will introduce some notation. A *multiset* is a set that allows duplicate elements (e.g., $\{2, 2, 2, 3, 3, 4, 5\}$ is a multiset with duplicate elements 2 and 3). If $X = \{x_1 = 0, x_2, \ldots, x_n\}$ is a set of n points on a line segment in increasing order, then ΔX denotes the multiset of *all* $\binom{n}{2}$ pairwise distances[3] between points in X:

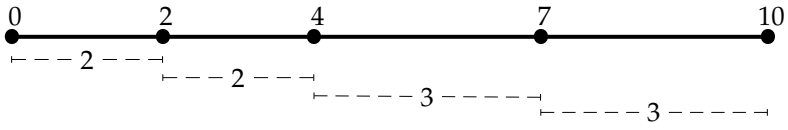
$$\Delta X = \{x_j - x_i : \ 1 \leq i < j \leq n\}.$$

For example, if X=$\{0, 2, 4, 7, 10\}$, then ΔX=$\{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$, which are the ten pairwise distances between these points (table 4.1). In restriction mapping, we are given ΔX, the experimental data about fragment lengths. The problem is to reconstruct X from ΔX. For example, could you infer

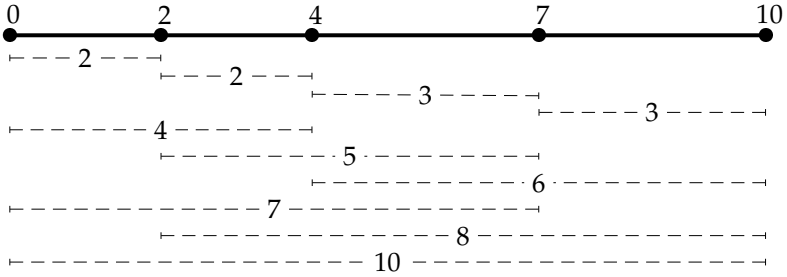
1. While restriction maps were popular research tools in the late 1980s, their role has been somewhat reduced in the last ten to fifteen years with the development of efficient DNA sequencing technologies. Though biologists rarely have to solve the DNA mapping problems in current research, we present them here as illustrations of branch-and-bound techniques for algorithm development.

2. Biologists typically work with billions of identical DNA molecules in solution. A complete digest corresponds to experimental conditions under which *every* DNA molecule at *every* restriction site is cut (i.e., the probability of cut at every restriction site is 1). Every linear DNA molecule with n restriction sites is cut into $n+1$ fragments that are recorded by gel electrophoresis as in figure 4.1 (a). A partial digest corresponds to experimental conditions that cut every DNA molecule at a given restriction site with probability less than 1. As a result, with some probability, the interval between any two (not necessarily consecutive) sites remains uncut, thus generating all fragments shown in figure 4.1 (b).

3. The notation $\binom{n}{k}$, read "n choose k," means "the number of distinct subsets of k elements taken from a (larger) set of n elements," and is given by the expression $\frac{n!}{(n-k)!k!}$. In particular, $\binom{n}{2} = \frac{n(n-1)}{2}$ is the number of different pairs of elements from an n-element set. For example, if $n = 5$, the set $\{1, 2, 3, 4, 5\}$ has $\binom{5}{2} = 10$ subsets formed by two elements: $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{1, 5\}$, $\{2, 3\}$, $\{2, 4\}$, $\{2, 5\}$, $\{3, 4\}$, $\{3, 5\}$, and $\{4, 5\}$. These ten subsets give rise to the elements $x_2 - x_1$, $x_3 - x_1$, $x_4 - x_1$, $x_5 - x_1$, $x_3 - x_2$, $x_4 - x_2$, $x_5 - x_2$, $x_4 - x_3$, $x_5 - x_3$, and $x_5 - x_4$.



(a) Complete digest.



(b) Partial digest.

Figure 4.1 Different methods of digesting a DNA molecule. A complete digest produces only fragments between consecutive restriction sites, while a partial digest yields fragments between any two restriction sites. Each of the dots represents a restriction site.

that the set $\Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$ was derived from $\{0, 2, 4, 7, 10\}$? Though gel electrophoresis allows one to determine the lengths of DNA fragments easily, it is often difficult to judge their multiplicity. That is, the number of different fragments of a given length can be difficult to determine. However, it is experimentally possible to do so with a lot of work, and we assume for the sake of simplifing the problem that this information is given to us as an input to the problem.

Table 4.1 Representation of $\Delta X = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$ as a two-dimensional table, with the elements of $X = \{0, 2, 4, 7, 10\}$ along both the top and right side. The element at (i, j) in the table is the value $x_j - x_i$ for $1 \leq i < j \leq n$.

	0	2	4	7	10
0		2	4	7	10
2			2	5	8
4				3	6
7					3
10					

Partial Digest Problem:
Given all pairwise distances between points on a line, reconstruct the positions of those points.

Input: The multiset of pairwise distances L, containing $\binom{n}{2}$ integers.

Output: A set X, of n integers, such that $\Delta X = L$

This *Partial Digest problem*, or PDP, is sometimes called the *Turnpike problem* in computer science. Suppose you knew the set of distances between every (not necessarily consecutive) pair of exits on a highway leading from one town to another. Could you reconstruct the geography of the highway from this information? That is, could you find the distance from the first town to each exit? Here, the "highway exits" are the restrictions sites in DNA; the lengths of the resulting DNA restriction fragments correspond to distances between highway exits. Computationally, the only difference between the Turnpike problem and the PDP is that the distances between exits are given in miles in the Turnpike problem, while the distances between restriction sites are given in nucleotides in the PDP.

We remark that it is not always possible to uniquely reconstruct a set X based only on ΔX. For example, for any integer v and set A, one can see that ΔA is equal to $\Delta(A \oplus \{v\})$, where $A \oplus \{v\}$ is defined to be $\{a + v : a \in A\}$, a *shift* of every element in A by v. Also $\Delta A = \Delta(-A)$, where $-A = \{-a : a \in A\}$ is the *reflection* of A. For example, sets $A = \{0, 2, 4, 7, 10\}$, $\Delta(A \oplus \{100\}) = \{100, 102, 104, 107, 110\}$, and $-A = \{-10, -7, -4, -2, 0\}$ all produce the same partial digest. The sets $\{0, 1, 3, 8, 9, 11, 12, 13, 15\}$ and $\{0, 1, 3, 4, 5, 7, 12, 13, 15\}$

present a less trivial example of this problem of nonuniqueness. The partial digests of these two sets is the same multiset of 36 elements[4]:
$\{1_4, 2_4, 3_4, 4_3, 5_2, 6_2, 7_2, 8_3, 9_2, 10_2, 11_2, 12_3, 13, 14, 15\}$.

	0	1	3	4	5	7	12	13	15
0		1	3	4	5	7	12	13	15
1			2	3	4	6	11	12	14
3				1	2	4	9	10	12
4					1	3	8	9	11
5						2	7	8	10
7							5	6	8
12								1	3
13									2
15									

	0	1	3	8	9	11	12	13	15
0		1	3	8	9	11	12	13	15
1			2	7	8	10	11	12	14
3				5	6	8	9	10	12
8					1	3	4	5	7
9						2	3	4	6
11							1	2	4
12								1	3
13									2
15									

In general, sets A and B are said to be *homometric* if $\Delta A = \Delta B$. Let U and V be two sets of numbers. One can verify that the multisets

$$U \oplus V = \{u + v : u \in U, v \in V\}$$

and

$$U \ominus V = \{u - v : u \in U, v \in V\}$$

are homometric (a problem at the end of this chapter). The "nontrivial" nine-point example above came from $U = \{6, 7, 9\}$ and $V = \{-6, 2, 6\}$. Indeed $U \oplus V = \{0, 1, 3, 8, 9, 11, 12, 13, 15\}$ while $U \ominus V = \{0, 1, 3, 4, 5, 7, 12, 13, 15\}$ as illustrated below:

$U \oplus V$	-6	2	6
6	0	8	12
7	1	9	13
9	3	11	15

$U \ominus V$	-6	2	6
6	12	4	0
7	13	5	1
9	15	7	3

While the PDP is to find one set X such that $\Delta X = L$, biologists are often interested in *all* homometric sets.

4.2 Impractical Restriction Mapping Algorithms

The algorithm below, BRUTEFORCEPDP, takes the list L of $\binom{n}{2}$ integers as an input, and returns the set X of n integers such that $\Delta X = L$. We remind the reader that we do not always provide complete details for all of the operations in pseudocode. In particular, we have not provided any subroutine to calculate ΔX; problem 4.1 asks you to do fill in the details for this step.

4. The notation 1_4 means that element 1 is repeated four times in this multiset.

```
BRUTEFORCEPDP(L, n)
1  M ← maximum element in L
2  for every set of n − 2 integers 0 < x_2 < ··· < x_{n−1} < M
3       X ← {0, x_2, ..., x_{n−1}, M}
4       Form ΔX from X
5       if ΔX = L
6            return X
7  output "No Solution"
```

BRUTEFORCEPDP is slow since it examines $\binom{M-1}{n-2}$ different sets of positions, which requires about $O(M^{n-2})$ time.[5]

One might question the wisdom of selecting $n-2$ *arbitrary* integers from the interval 0 to M. For example, if L does not contain the number 5, there is really no point in choosing any $x_i = 5$, though the above algorithm will do so. Indeed, observing that all points in X have to correspond to *some* distance in ΔX, we can select $n-2$ distinct elements from L rather than the less constrained selection from the interval $(0, M)$. Since M may be large, even with a small number of points, building a new algorithm that makes choices of x_i based only on elements in L yields an improvement in efficiency.

```
ANOTHERBRUTEFORCEPDP(L, n)
1  M ← maximum element in L
2  for every set of n − 2 integers 0 < x_2 < ··· < x_{n−1} < M from L
3       X ← {0, x_2, ..., x_{n−1}, M}
4       Form ΔX from X
5       if ΔX = L
6            return X
7  output "No Solution"
```

This algorithm examines $\binom{|L|}{n-2}$ different sets of integers, but $|L| = \frac{n(n-1)}{2}$, so ANOTHERBRUTEFORCEPDP takes roughly $O(n^{2n-4})$ time. This is still not practical, but since M can be arbitrarily large compared to n, this is actually a more efficient algorithm than BRUTEFORCEPDP. For example, BRUTEFORCEPDP takes a very long time to execute when called on an input of $L = \{2, 998, 1000\}$, but ANOTHERBRUTEFORCEPDP takes very little time. Here, n is 3 while M is 1000.

5. Although a careful mathematical analysis of the running time leads to a somewhat smaller number, it does not help much in practice.

4.3 A Practical Restriction Mapping Algorithm

In 1990, Steven Skiena described a different brute force algorithm for the PDP that works well in practice.[6] First, find the largest distance in L; this must determine the two outermost points of X, and we can delete this distance from L. Now that we have fixed the two outermost points, we select the largest remaining distance in L, call it δ. One of the points that generated δ must be one of the two outermost points, since δ is the *largest* remaining distance; thus, we have one of two choices to place a point: δ from the leftmost point, or δ from the rightmost point. Suppose we decide to include the point that is δ from the leftmost point in the set. We can calculate the pairwise distances between this new position and all the other positions that we have chosen, and ask if these distances are in L. If so, then we remove those distances from L and repeat by selecting the next largest remaining distance in L and so on. If these pairwise distances are not in L, then we choose the position that is δ from the rightmost point, and perform the same query. However, if these pairwise distances are not in L either, then we must have made a bad choice at some earlier step and we need to *backtrack* a few steps, reverse a decision, and try again. If we ever get to the point where L is empty, then we have found a valid solution.[7]

For example, suppose $L = \{2, 2, 3, 3, 4, 5, 6, 7, 8, 10\}$. The size of L is $\binom{n}{2} = \frac{n(n-1)}{2} = 10$, where n is the number of points in the solution. In this case, n must be 5, and we will refer to the positions in X as $x_1 = 0$, x_2, x_3, x_4 and x_5, from left to right, on the line.

Since 10 is the largest distance in L, x_5 must be at position 10, so we remove this distance $x_5 - x_1 = 10$ from L to obtain

$$X = \{0, 10\} \qquad L = \{2, 2, 3, 3, 4, 5, 6, 7, 8\} \ .$$

The largest remaining distance is 8. We have two choices: either $x_4 = 8$ or $x_2 = 2$. Since those two cases are mirror images of each other, without loss of generality, we can assume $x_2 = 2$. After removal of distances $x_5 - x_2 = 8$ and $x_2 - x_1 = 2$ from L, we obtain

$$X = \{0, 2, 10\} \qquad L = \{2, 3, 3, 4, 5, 6, 7\} \ .$$

6. To be more accurate, this algorithm works well for high-quality PDP data. However, the bottleneck in the PDP technique is the difficulty in acquiring accurate data.
7. This is an example of a cookbook description; the pseudocode of this algorithm is described below.

Now 7 is the largest remaining distance, so either $x_4 = 7$ or $x_3 = 3$. If $x_3 = 3$, then $x_3 - x_2 = 1$ must be in L, but it is not, so x_4 must be at 7. After removing distances $x_5 - x_4 = 3$, $x_4 - x_2 = 5$, and $x_4 - x_1 = 7$ from L, we obtain

$$X = \{0, 2, 7, 10\} \qquad L = \{2, 3, 4, 6\} \; .$$

Now 6 is the largest remaining distance, and we once again have only two choices: either $x_3 = 4$ or $x_3 = 6$. If $x_3 = 6$, the distance $x_4 - x_3 = 1$ must be in L, but it is not. We are left with only one choice, $x_3 = 4$, and this provides a solution $X = \{0, 2, 4, 7, 10\}$ to the PDP.

The PARTIALDIGEST algorithm shown below works with the list of pairwise distances, L, and uses the function DELETE(y, L) which removes the value y from L. We use the notation $\Delta(y, X)$ to denote the multiset of distances between a point y and all points in a set X. For example,

$$\Delta(2, \{1, 3, 4, 5\}) = \{1, 1, 2, 3\} \, .$$

```
PARTIALDIGEST(L)
 1  width ← Maximum element in L
 2  DELETE(width, L)
 3  X ← {0, width}
 4  PLACE(L, X)

PLACE(L, X)
 1  if L is empty
 2      output X
 3      return
 4  y ← Maximum element in L
 5  if Δ(y, X) ⊆ L
 6      Add y to X and remove lengths Δ(y, X) from L
 7      PLACE(L, X)
 8      Remove y from X and add lengths Δ(y, X) to L
 9  if Δ(width − y, X) ⊆ L
10      Add width − y to X and remove lengths Δ(width − y, X) from L
11      PLACE(L, X)
12      Remove width − y from X and add lengths Δ(width − y, X) to L
13  return
```

After each recursive call in PLACE, we undo our modifications to the sets X and L in order to restore them for the next recursive call. It is important to note that this algorithm will list *all* sets X with $\Delta X = L$.

At first glance, this algorithm looks efficient—at each point we examine two alternatives ("left" or "right"), ruling out the obviously incorrect decisions that lead to inconsistent distances. Indeed, this algorithm is very fast

for most instances of the PDP since usually only one of the two alternatives, "left" or "right," is viable at any step. It was not clear for a number of years whether or not this algorithm is polynomial in the worst case—sometimes both alternatives are viable. If both "left" and "right" alternatives hold and if this continues to happen in future steps of the algorithm, then the performance of the algorithm starts growing as 2^k where k is the number of such "ambiguous" steps.[8]

Let $T(n)$ be the maximum time PARTIALDIGEST takes to find the solution for an n-point instance of the PDP. If there is only one viable alternative at every step, then PARTIALDIGEST steadily reduces the size of the problem by one and calls itself recursively, so

$$T(n) = T(n-1) + O(n),$$

where $O(n)$ is the work spent adjusting the sets X and L. However, if there are two alternatives, then

$$T(n) = 2T(n-1) + O(n).$$

While the expressions $T(n) = T(n-1) + O(n)$ and $T(n) = 2T(n-1) + O(n)$ bear a superficial similarity in form, they each lead to very different expressions for the algorithm's running time. One is quadratic, as we saw when analyzing SELECTIONSORT, and the other exponential, as we saw with HANOITOWERS. In fact, polynomial algorithms for the PDP were unknown until 2002 when Maurice Nivat and colleagues designed the first one.

4.4 Regulatory Motifs in DNA Sequences

Fruit flies, like humans, are susceptible to infections from bacteria and other pathogens. Although fruit flies do not have as sophisticated an immune system as humans do, they have a small set of *immunity genes* that are usually dormant in the fly genome, but somehow get switched on when the organism gets infected. When these genes are turned on, they produce proteins that destroy the pathogen, usually curing the infection.

One could design an experiment that is rather unpleasant to the flies, but very informative to biologists: infect flies with a bacterium, then grind up the flies and measure (perhaps with a DNA array) which genes are switched on

8. There exist pathological examples forcing the algorithm to explore *both* "left" and "right" alternatives at nearly every step.