# 10 *Clustering and Trees*

A common problem in biology is to partition a set of experimental data into groups (clusters) in such a way that the data points within the same cluster are highly similar while data points in different clusters are very different. This problem is far from simple, and this chapter covers several algorithms that perform different types of clustering. There is no simple recipe for choosing one particular approach over another for a particular clustering problem, just as there is no universal notion of what constitutes a "good cluster." Nonetheless, these algorithms can yield significant insight into data and allow one, for example, to identify clusters of genes with similar functions even when it is not clear what particular role these genes play. We conclude this chapter with studies of evolutionary tree reconstruction, which is closely related to clustering.

## 10.1 Gene Expression Analysis

Sequence comparison often helps to discover the function of a newly sequenced gene by finding similarities between the new gene and previously sequenced genes with known functions. However, for many genes, the sequence similarity of genes in a functional family is so weak that one cannot reliably derive the function of the newly sequenced gene based on sequence alone. Moreover, genes with the same function sometimes have no sequence similarity at all. As a result, the functions of more than 40% of the genes in sequenced genomes are still unknown.

In the last decade, a new approach to analyzing gene functions has emerged. DNA arrays allow one to analyze the *expression levels* (amount of mRNA produced in the cell) of many genes under many time points and conditions and to reveal which genes are switched on and switched off in the cell.[1] The outcome of this type of study is an $n \times m$ *expression matrix* $\mathbf{I}$, with the $n$ rows corresponding to genes, and the $m$ columns corresponding to different time points and different conditions. The expression matrix $\mathbf{I}$ represents *intensities* of hybridization signals as provided by a DNA array. In reality, expression matrices usually represent transformed and normalized intensities rather than the raw intensities obtained as a result of a DNA array experiment, but we will not discuss this transformation.

The element $I_{i,j}$ of the expression matrix represents the expression level of gene $i$ in experiment $j$; the entire $i$th row of the expression matrix is called the *expression pattern* of gene $i$. One can look for pairs of genes in an expression matrix with similar expression patterns, which would be manifested as two similar rows. Therefore, if the expression patterns of two genes are similar, there is a good chance that these genes are somehow related, that is, they either perform similar functions or are involved in the same biological process.[2] Accordingly, if the expression pattern of a newly sequenced gene is similar to the expression pattern of a gene with known function, a biologist may have reason to suspect that these genes perform similar or related functions. Another important application of expression analysis is in the deciphering of regulatory pathways; similar expression patterns usually imply coregulation. However, expression analysis should be done with caution since DNA arrays typically produce noisy data with high error rates.

*Clustering algorithms* group genes with similar expression patterns into *clusters* with the hope that these clusters correspond to groups of functionally related genes. To cluster the expression data, the $n \times m$ expression matrix is often transformed into an $n \times n$ *distance matrix* $\mathbf{d} = (d_{i,j})$ where $d_{i,j}$ reflects how similar the expression patterns of genes $i$ and $j$ are (see figure 10.1).[3] The goal of clustering is to group genes into clusters satisfying the following two conditions:

- *Homogeneity.* Genes (rather, their expression patterns) within a cluster

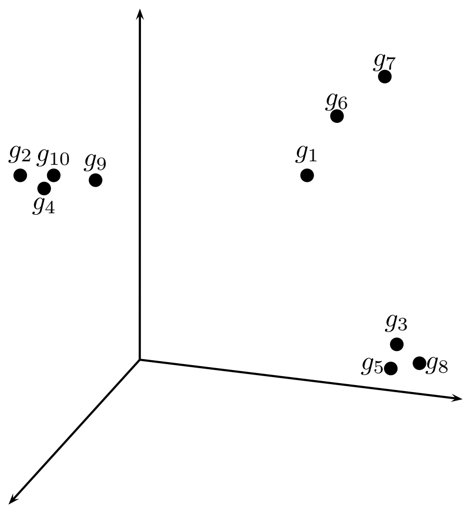
1. Expression analysis studies implicitly assume that the amount of mRNA (as measured by a DNA array) is correlated with the amount of its protein produced by the cell. We emphasize that a number of processes affect the production of proteins in the cell (transcription, splicing, translation, post-translational modifications, protein degradation, etc.) and therefore this correlation may not be straightforward, but it is still significant.
2. Of course, we do not exclude the possibility that expression patterns of two genes may look similar simply by chance, but the probability of such a chance similarity decreases as we increase the number of time points.
3. The distance matrix is typically larger than the expression matrix since $n \gg m$ in most expression studies.

| Time | 1 hr | 2 hr | 3 hr |
|---|---|---|---|
| $g_1$ | 10.0 | 8.0 | 10.0 |
| $g_2$ | 10.0 | 0.0 | 9.0 |
| $g_3$ | 4.0 | 8.5 | 3.0 |
| $g_4$ | 9.5 | 0.5 | 8.5 |
| $g_5$ | 4.5 | 8.5 | 2.5 |
| $g_6$ | 10.5 | 9.0 | 12.0 |
| $g_7$ | 5.0 | 8.5 | 11.0 |
| $g_8$ | 2.7 | 8.7 | 2.0 |
| $g_9$ | 9.7 | 2.0 | 9.0 |
| $g_{10}$ | 10.2 | 1.0 | 9.2 |

(a) Intensity matrix, **I**

| | $g_1$ | $g_2$ | $g_3$ | $g_4$ | $g_5$ | $g_6$ | $g_7$ | $g_8$ | $g_9$ | $g_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $g_1$ | 0.0 | 8.1 | 9.2 | 7.7 | 9.3 | 2.3 | 5.1 | 10.2 | 6.1 | 7.0 |
| $g_2$ | 8.1 | 0.0 | 12.0 | 0.9 | 12.0 | 9.5 | 10.1 | 12.8 | 2.0 | 1.0 |
| $g_3$ | 9.2 | 12.0 | 0.0 | 11.2 | 0.7 | 11.1 | 8.1 | 1.1 | 10.5 | 11.5 |
| $g_4$ | 7.7 | 0.9 | 11.2 | 0.0 | 11.2 | 9.2 | 9.5 | 12.0 | 1.6 | 1.1 |
| $g_5$ | 9.3 | 12.0 | 0.7 | 11.2 | 0.0 | 11.2 | 8.5 | 1.0 | 10.6 | 11.6 |
| $g_6$ | 2.3 | 9.5 | 11.1 | 9.2 | 11.2 | 0.0 | 5.6 | 12.1 | 7.7 | 8.5 |
| $g_7$ | 5.1 | 10.1 | 8.1 | 9.5 | 8.5 | 5.6 | 0.0 | 9.1 | 8.3 | 9.3 |
| $g_8$ | 10.2 | 12.8 | 1.1 | 12.0 | 1.0 | 12.1 | 9.1 | 0.0 | 11.4 | 12.4 |
| $g_9$ | 6.1 | 2.0 | 10.5 | 1.6 | 10.6 | 7.7 | 8.3 | 11.4 | 0.0 | 1.1 |
| $g_{10}$ | 7.0 | 1.0 | 11.5 | 1.1 | 11.6 | 8.5 | 9.3 | 12.4 | 1.1 | 0.0 |

(b) Distance matrix, **d**




(c) Expression patterns as points in three-dimentsional space.

**Figure 10.1** An "expression" matrix of ten genes measured at three time points, and the corresponding distance matrix. Distances are calculated as the Euclidean distance in three-dimensional space.

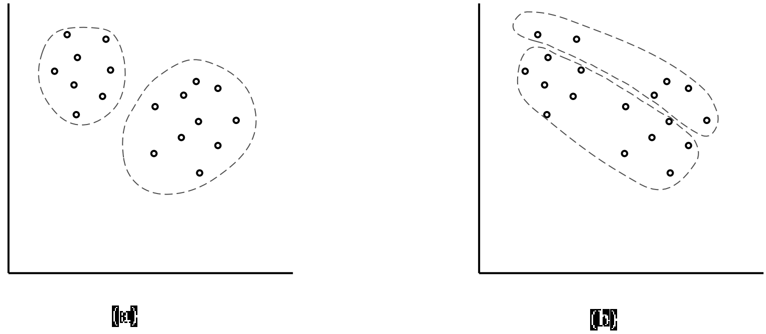


**Figure 10.2** Data can be grouped into clusters. Some clusters are better than others: the two clusters in a) exhibit good homogeneity and separation, while the clusters in b) do not.

should be highly similar to each other. That is, $d_{i,j}$ should be small if $i$ and $j$ belong to the same cluster.

- *Separation.* Genes from different clusters should be very different. That is, $d_{i,j}$ should be large if $i$ and $j$ belong to different clusters.

An example of clustering is shown in figure 10.2. Figure 10.2 (a) shows a good partition according to the above two properties, while (b) shows a bad one. Clustering algorithms try to find a good partition.

A "good" clustering of data is one that adheres to these goals. While we hope that a better clustering of genes gives rise to a better grouping of genes on a functional level, the final analysis of resulting clusters is left to biologists.

Different tissues express different genes, and there are typically over 10,000 genes expressed in any one tissue. Since there are about 100 different tissue types, and since expression levels are often measured over many time points, gene expression experiments can generate vast amounts of data which can be hard to interpret. Compounding these difficulties, expression levels of related genes may vary by several orders of magnitude, thus creating the problem of achieving accurate measurements over a large range of expression levels; genes with low expression levels may be related to genes with high expression levels.

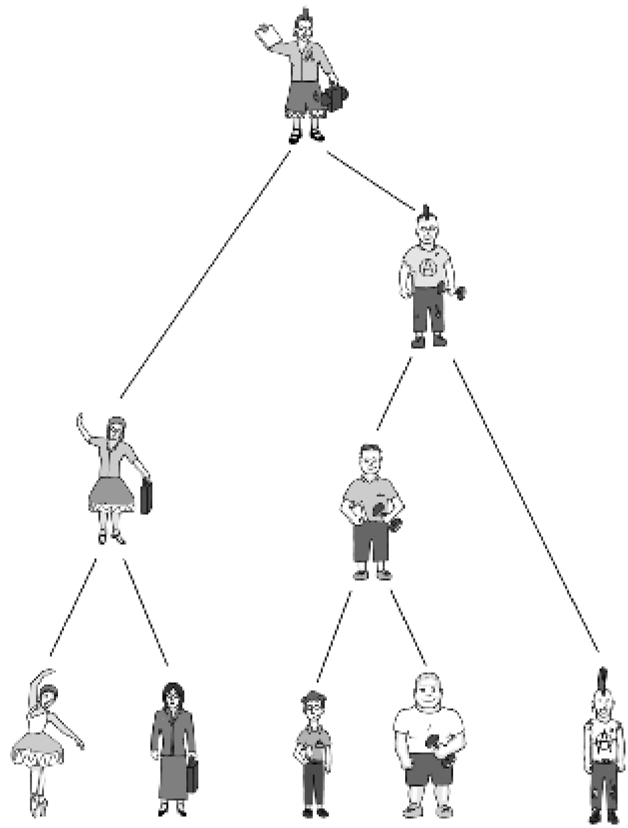


**Figure 10.3** A schematic of hierarchical clustering.

## 10.2 Hierarchical Clustering

In many cases clusters have subclusters, these have subsubclusters, and so on. For example, mammals can be broken down into primates, carnivora, bats, marsupials, and many other orders. The order carnivora can be further broken down into cats, hyenas, bears, seals, and many others. Finally, cats can be broken into thirty seven species.[4]

4. Lions, tigers, leopards, jaguars, lynx, cheetahs, pumas, golden cats, domestic cats, small wildcats, ocelots, and many others.

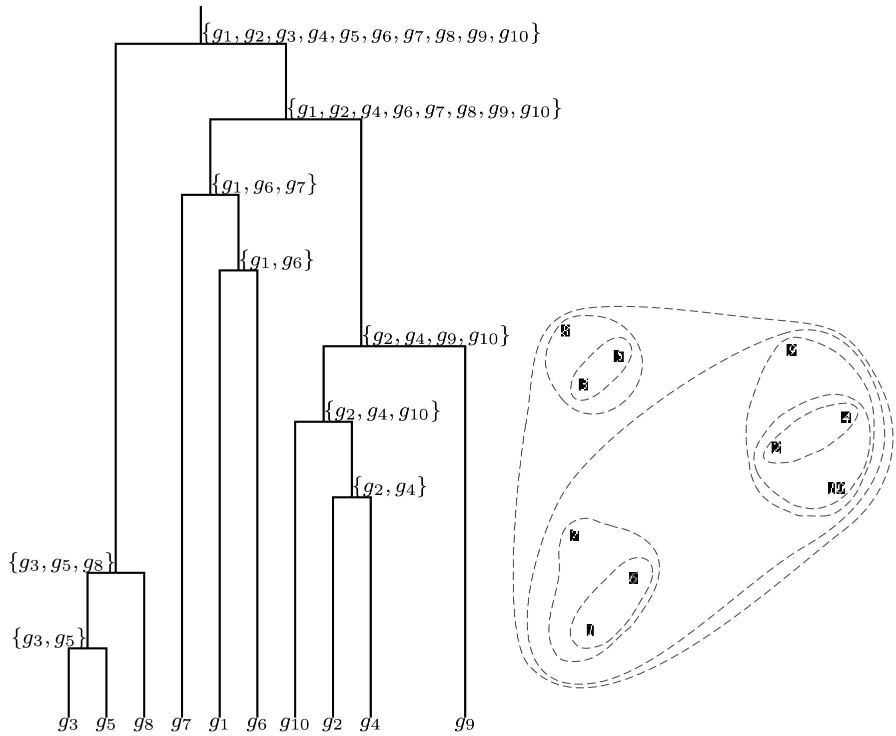



**Figure 10.4** A hierarchical clustering of the data in figure 10.1

*Hierarchical clustering* (fig. 10.3) is a technique that organizes elements into a *tree*, rather than forming an explicit partitioning of the elements into clusters. In this case, the genes are represented as the leaves of a tree. The edges of the trees are assigned *lengths* and the distances between leaves—that is, the length of the path in the tree that connects two leaves—correlate with entries in the distance matrix. Such trees are used in both the analysis of expression data and in studies of molecular evolution which we will discuss below.

Figure 10.4 shows a tree that represents clustering of the data in figure 10.1. This tree actually describes a family of different partitions into clusters, each with a different number of clusters (one for each value from $1$ to $n$). You can see what these partitions by drawing a horizontal line through the tree. Each line crosses the tree at $i$ points ($1 \leq i \leq k$) and correspond to $i$ clusters.

The HIERARCHICALCLUSTERING algorithm below takes an $n \times n$ distance matrix $\mathbf{d}$ as an input, and progressively generates $n$ different partitions of the data as the tree it outputs. The largest partition has $n$ single-element clusters, with every element forming its own cluster. The second-largest partition combines the two closest clusters from the largest partition, and thus has $n-1$ clusters. In general, the $i$th partition combines the two closest clusters from the $(i-1)$th partition and has $n-i+1$ clusters.

HIERARCHICALCLUSTERING($\mathbf{d}, n$)
1 Form $n$ clusters, each with $1$ element
2 Construct a graph $T$ by assigning an isolated vertex to each cluster
3 **while** there is more than $1$ cluster
4     Find the two closest clusters $C_1$ and $C_2$
5     Merge $C_1$ and $C_2$ into new cluster $C$ with $|C_1| + |C_2|$ elements
6     Compute distance from $C$ to all other clusters
7     Add a new vertex $C$ to $T$ and connect to vertices $C_1$ and $C_2$
8     Remove rows and columns of $\mathbf{d}$ corresponding to $C_1$ and $C_2$
9     Add a row and column to $\mathbf{d}$ for the new cluster $C$
10 **return** $T$

Line $6$ in the algorithm is (intentionally) left ambiguous; clustering algorithms vary in how they compute the distance between the newly formed cluster and any other cluster. Different formulas for recomputing distances yield different answers from the same hierarchical clustering algorithm. For example, one can define the distance between two clusters as the smallest distance between any pair of their elements

$$d_{min}(C^*, C) = \min_{x \in C^*, y \in C} d(x, y)$$

or the average distance between their elements

$$d_{avg}(C^*, C) = \frac{1}{|C^*||C|} \sum_{x \in C^*, y \in C} d(x, y).$$

Another distance function estimates distance based on the separation of $C_1$ and $C_2$ in HIERARCHICALCLUSTERING:

$$d(C^*, C) = \frac{d(C^*, C_1) + d(C^*, C_2) - d(C_1, C_2)}{2}$$

In one of the first expression analysis studies, Michael Eisen and colleagues used hierarchical clustering to analyze the expression profiles of $8600$ genes over thirteen time points to find the genes responsible for the growth response of starved human cells. The HIERARCHICALCLUSTERING resulted in a tree consisting of five main subtrees and many smaller subtrees. The genes within these five clusters had similar functions, thus confirming that the resulting clusters are biologically sensible.

## 10.3 *k*-Means Clustering

One can view the $n$ rows of the $n \times m$ expression matrix as a set of $n$ points in $m$-dimensional space and partition them into $k$ subsets, pretending that $k$—the number of clusters—is known in advance.

One of the most popular clustering methods for points in multidimensional spaces is called *$k$-Means clustering*. Given a set of $n$ data points in $m$-dimensional space and an integer $k$, the problem is to determine a set of $k$ points, or *centers*, in $m$-dimensional space that minimize the *squared error distortion* defined below. Given a data point $v$ and a set of $k$ centers $\mathcal{X} = \{x_1, \ldots x_k\}$, define the distance from $v$ to the centers $\mathcal{X}$ as the distance from $v$ to the closest point in $\mathcal{X}$, that is, $d(v, \mathcal{X}) = \min_{1 \le i \le k} d(v, x_i)$. We will assume for now that $d(v, x_i)$ is just the Euclidean[5] distance in $m$ dimensions. The squared error distortion for a set of $n$ points $\mathcal{V} = \{v_1, \ldots v_n\}$, and a set of $k$ centers $\mathcal{X} = \{x_1, \ldots x_k\}$, is defined as the mean squared distance from each data point to its nearest center:

$$d(\mathcal{V}, \mathcal{X}) = \frac{\sum_{i=1}^{n} d(v_i, \mathcal{X})^2}{n}$$

---

**$k$-Means Clustering Problem**:
*Given $n$ data points, find $k$ center points minimizing the squared error distortion.*

**Input:** A set, $\mathcal{V}$, consisting of $n$ points and a parameter $k$.

**Output:** A set $\mathcal{X}$ consisting of $k$ points (called centers) that minimizes $d(\mathcal{V}, \mathcal{X})$ over all possible choices of $\mathcal{X}$.

---

5. Chapter 2 contains a sample algorithm to calculate this when $m$ is 2.

While the above formulation does not explicitly address *clustering* $n$ points into $k$ clusters, a clustering can be obtained by simply assigning each point to its closest center. Although the $k$-Means Clustering problem looks relatively simple, there are no efficient (polynomial) algorithms known for it. The *Lloyd* $k$-Means clustering algorithm is one of the most popular clustering heuristics that often generates good solutions in gene expression analysis. The Lloyd algorithm randomly selects an arbitrary partition of points into $k$ clusters and tries to improve this partition by moving some points between clusters. In the beginning one can choose arbitrary $k$ points as "cluster representatives." The algorithm iteratively performs the following two steps until either it converges or until the fluctuations become very small:

- Assign each data point to the cluster $C_i$ corresponding to the closest cluster representative $x_i$ ($1 \le i \le k$)
- After the assignments of all $n$ data points, compute new cluster representatives according to the center of gravity of each cluster, that is, the new cluster representative is $\frac{\sum_{v \in C} v}{|C|}$ for every cluster $C$.

The Lloyd algorithm often converges to a local minimum of the squared error distortion function rather than the global minimum. Unfortunately, interesting objective functions other than the squared error distortion lead to similarly difficult problems. For example, finding a good clustering can be quite difficult if, instead of the squared error distortion ($\sum_{i=1}^{n} d(v_i, \mathcal{X})^2$), one tries to minimize $\sum_{i=1}^{n} d(v_i, \mathcal{X})$ (*k-Median problem*) or $\max_{1 \le i \le n} d(v_i, \mathcal{X})$ (*k-Center problem*). We remark that all of these definitions of clustering cost emphasize the homogeneity condition and more or less ignore the other important goal of clustering, the separation condition. Moreover, in some unlucky instances of the $k$-Means Clustering problem, the algorithm may converge to a local minimum that is *arbitrarily* bad compared to an optimal solution (a problem at the end of this chapter).

While the Lloyd algorithm is very fast, it can significantly rearrange every cluster in every iteration. A more conservative approach is to move only one element between clusters in each iteration. We assume that every partition $P$ of the $n$-element set into $k$ clusters has an associated *clustering cost*, denoted $cost(P)$, that measures the quality of the partition $P$: the smaller the clustering cost of a partition, the better that clustering is.[6] The squared error distortion is one particular choice of $cost(P)$ and assumes that each center

6. "Better" here is better clustering, not a better biological grouping of genes.

point is the center of gravity of its cluster. The pseudocode below implicitly assumes that $cost(P)$ can be efficiently computed based either on the distance matrix or on the expression matrix. Given a partition $P$, a cluster $C$ within this partition, and an element $i$ outside $C$, $P_{i \to C}$ denotes the partition obtained from $P$ by moving the element $i$ from its cluster to $C$. This move improves the clustering cost only if $\Delta(i \to C) = cost(P) - cost(P_{i \to C}) > 0$, and the PROGRESSIVEGREEDYK-MEANS algorithm searches for the "best" move in each step (i.e., a move that maximizes $\Delta(i \to C)$ for all $C$ and for all $i \notin C$).

```
PROGRESSIVEGREEDYK-MEANS(k)
1   Select an arbitrary partition P into k clusters.
2   while forever
3       bestChange ← 0
4       for every cluster C
5           for every element i ∉ C
6               if moving i to cluster C reduces the clustering cost
7                   if Δ(i → C) > bestChange
8                       bestChange ← Δ(i → C)
9                       i* ← i
10                      C* ← C
11      if bestChange > 0
12          change partition P by moving i* to C*
13      else
14          return P
```

Even though line 2 makes an impression that this algorithm may loop endlessly, the return statement on line 14 saves us from an infinitely long wait. We stop iterating when no move allows for an improvement in the score; this eventually has to happen.[7]

## 10.4 Clustering and Corrupted Cliques

A complete graph, written $K_n$, is an (undirected) graph on $n$ vertices with every two vertices connected by an edge. A *clique graph* is a graph in which every connected component is a complete graph. Figure 10.5 (a) shows a

7. What would be the natural implication if there could always be an improvement in the score?

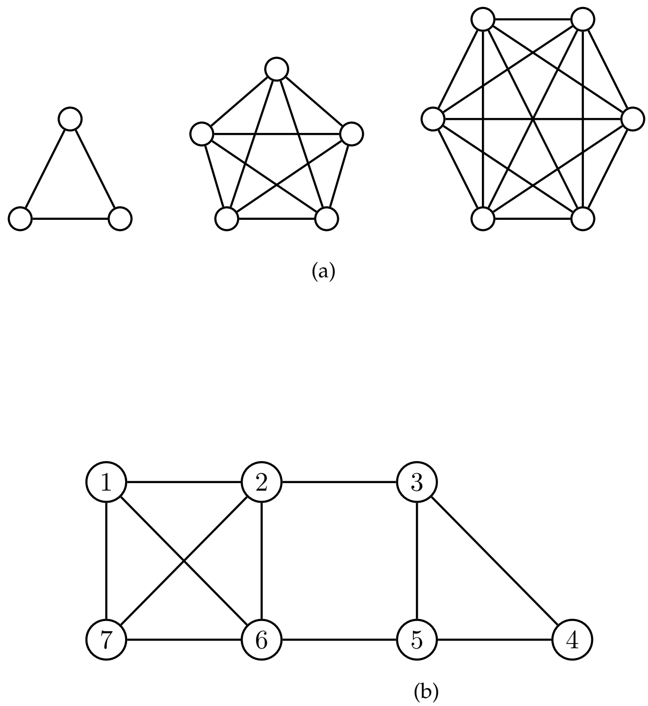



**Figure 10.5** a) A clique graph consisting of the three connected components $K_3$, $K_5$, and $K_6$. b) A graph with 7 vertices that has $4$ cliques formed by vertices $\{1, 2, 6, 7\}$, $\{2, 3\}$, $\{5, 6\}$, and $\{3, 4, 5\}$.

clique graph consisting of three connected components, $K_3$, $K_5$, and $K_6$. Every partition of $n$ elements into $k$ clusters can be represented by a clique graph on $n$ vertices with $k$ cliques. A subset of vertices $V' \subset V$ in a graph $G(V, E)$ forms a *complete subgraph* if the *induced subgraph* on these vertices is complete, that is, every two vertices $v$ and $w$ in $V'$ are connected by an edge in the graph. For example, vertices 1, 6, and 7 form a complete subgraph of the graph in figure 10.5 (b). A *clique* in the graph is a maximal complete subgraph, that is, a complete subgraph that is not contained inside any other complete subgraph. For example, in figure 10.5 (b), vertices $1$, $6$, and $7$ form a complete subgraph but do not form a clique, but vertices $1$, $2$, $6$, and $7$ do.

In expression analysis studies, the distance matrix $(d_{i,j})$ is often further transformed into a *distance graph* $G = G(\theta)$, where the vertices are genes and there is an edge between genes $i$ and $j$ if and only if the distance between them is below the threshold $\theta$, that is, if $d_{i,j} < \theta$. A clustering of genes that satisfies the homogeneity and separation principles for an appropriately chosen $\theta$ will correspond to a distance graph that is also a clique graph. However, errors in expression data, and the absence of a "universally good" threshold $\theta$ often results in distance graphs that do not quite form clique graphs (fig. 10.6). Some elements of the distance matrix may fall below the distance threshold for unrelated genes (adding edges between different clusters), while other elements of the distance matrix exceed the distance threshold for related genes (removing edges within clusters). Such erroneous edges corrupt the clique graph, raising the question of how to transform the distance graph into a clique graph using the smallest number of edge removals and edge additions.

---

**Corrupted Cliques Problem**:
*Determine the smallest number of edges that need to be added or removed to transform a graph into a clique graph.*

**Input:** A graph $G$.

**Output:** The smallest number of additions and removals of edges that will transform $G$ into a clique graph.

---

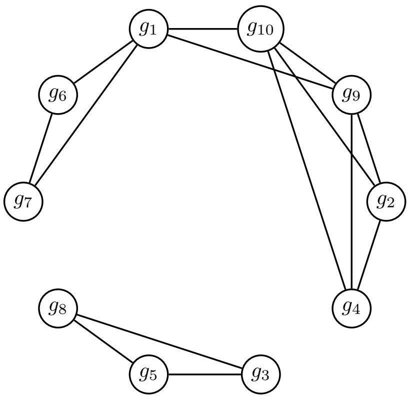
It turns out that the Corrupted Cliques problem is $\mathcal{NP}$-hard, so some heuristics have been proposed to approximately solve it. Below we describe the time-consuming PCC (*Parallel Classification with Cores*) algorithm, and the less theoretically sound, but practical, CAST (*Cluster Affinity Search Technique*) algorithm inspired by PCC.

Suppose we attempt to cluster a set of genes $S$, and suppose further that $S'$ is a subset of $S$. If we are somehow magically given the correct clustering[8] $\{C_1, \ldots, C_k\}$ of $S'$, could we extend this clustering of $S'$ into a clustering of the entire gene set $S$? Let $S \setminus S'$ be the set of unclustered genes, and $N(j, C_i)$ be the number of edges between gene $j \in S \setminus S'$ and genes from the cluster $C_i$ in the distance graph. We define the *affinity* of gene $j$ to cluster $C_i$ as $\frac{N(j,C_i)}{|C_i|}$.

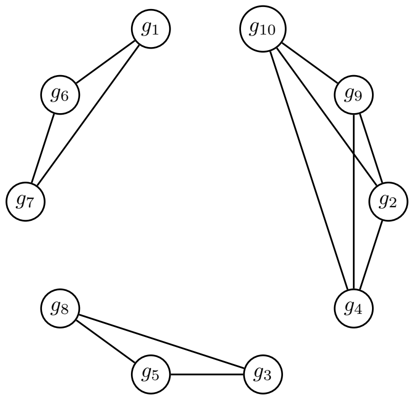
8. By the "correct clustering of $S'$," we mean the classification of elements of $S'$ (which is a subset of the entire gene set) into the same clusters as they would be in the clustering of $S$ that has the optimal clustering score.

| | $g_1$ | $g_2$ | $g_3$ | $g_4$ | $g_5$ | $g_6$ | $g_7$ | $g_8$ | $g_9$ | $g_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $g_1$ | 0.0 | 8.1 | 9.2 | 7.7 | 9.3 | **2.3** | **5.1** | 10.2 | **6.1** | **7.0** |
| $g_2$ | 8.1 | 0.0 | 12.0 | **0.9** | 12.0 | 9.5 | 10.1 | 12.8 | **2.0** | **1.0** |
| $g_3$ | 9.2 | 12.0 | 0.0 | 11.2 | **0.7** | 11.1 | 8.1 | **1.1** | 10.5 | 11.5 |
| $g_4$ | 7.7 | **0.9** | 11.2 | 0.0 | 11.2 | 9.2 | 9.5 | 12.0 | **1.6** | **1.1** |
| $g_5$ | 9.3 | 12.0 | **0.7** | 11.2 | 0.0 | 11.2 | 8.5 | **1.0** | 10.6 | 11.6 |
| $g_6$ | **2.3** | 9.5 | 11.1 | 9.2 | 11.2 | 0.0 | **5.6** | 12.1 | 7.7 | 8.5 |
| $g_7$ | **5.1** | 10.1 | 8.1 | 9.5 | 8.5 | **5.6** | 0.0 | 9.1 | 8.3 | 9.3 |
| $g_8$ | 10.2 | 12.8 | **1.1** | 12.0 | **1.0** | 12.1 | 9.1 | 0.0 | 11.4 | 12.4 |
| $g_9$ | **6.1** | **2.0** | 10.5 | **1.6** | 10.6 | 7.7 | 8.3 | 11.4 | 0.0 | **1.1** |
| $g_{10}$ | **7.0** | **1.0** | 11.5 | **1.1** | 11.6 | 8.5 | 9.3 | 12.4 | **1.1** | 0.0 |

(a) Distance matrix, $\mathbf{d}$ (distances shorter than 7 are shown in bold).




(b) Distance graph for $\theta = 7$.




(c) Clique graph.

**Figure 10.6** The distance graph (b) for $\theta = 7$ is not quite a clique graph. However, it can be transformed into a clique graph (c) by removing edges $(g_1, g_{10})$ and $(g_1, g_9)$.

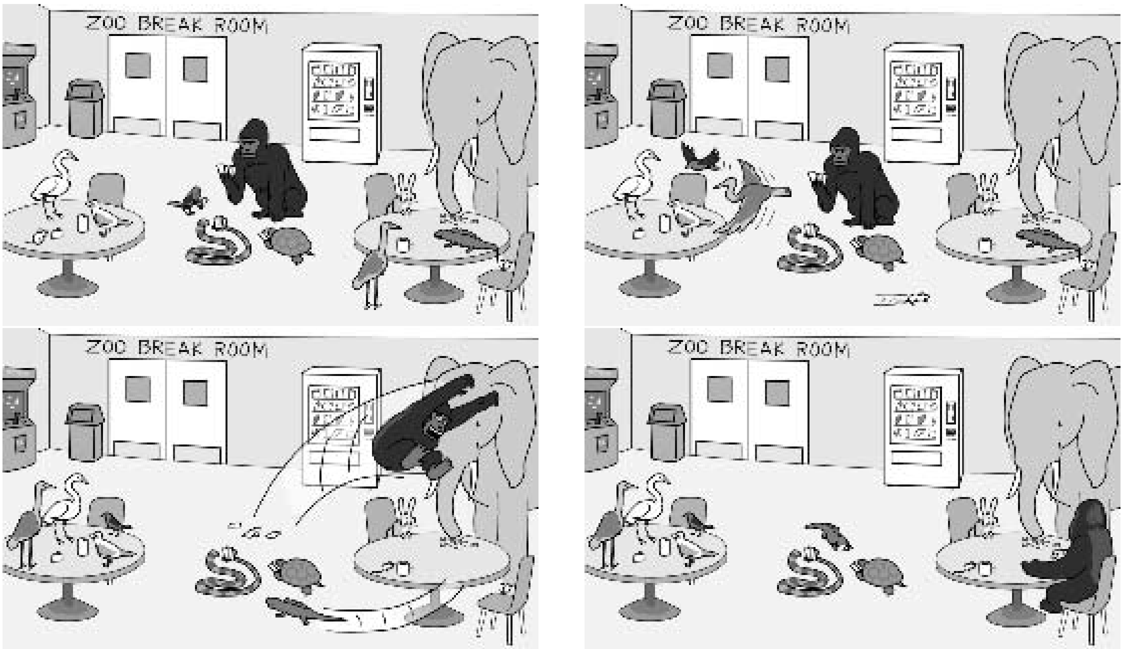
A natural *maximal affinity* approach would be to put every unclustered gene $j$ into the cluster $C_i$ with the highest affinity to $j$, that is, the cluster that maximizes $\frac{N(j,C_i)}{|C_i|}$. In this way, the clustering of $S'$ can be extended into clustering of the entire gene set $S$. In 1999, Amir Ben-Dor and colleagues developed the PCC clustering algorithm, which relies on the assumption that if $S'$ is sufficiently large and the clustering of $S'$ is correct, then the clustering of the entire gene set is likely to to be correct.

The only problem is that the correct clustering of $S'$ is unknown! The way around this is to generate *all* possible clusterings of $S'$, extend them into a clustering of the entire gene set $S$, and select the resulting clustering with the best clustering score. As attractive as it may sound, this is not practical (unless $S'$ is very small) since the number of possible partitions of a set $S'$ into $k$ clusters is $k^{|S'|}$. The PCC algorithm gets around this problem by making $S'$ extremely small and generating *all* partitions of this set into $k$ clusters. It then extends each of these $k^{|S'|}$ partitions into a partition of the entire $n$-element gene set by the two-stage procedure described below. The distance graph $G$ guides these extensions based on the maximal affinity approach. The function $score(P)$ is defined to be the number of edges necessary to add or remove to turn $G$ into a clique graph according to the partition $P$.[9] The PCC algorithm below clusters the set of elements $S$ into $k$ clusters according to the graph $G$ by extending partitions of subsets of $S$ using the maximal affinity approach:

PCC$(G, k)$<br>
$S \leftarrow$ set of vertices in the distance graph $G$<br>
$n \leftarrow$ number of elements in $S$<br>
$bestScore \leftarrow \infty$<br>
Randomly select a "very small" set $S' \subset S$, where $|S'| = \log \log n$<br>
Randomly select a "small" set $S'' \subset (S \setminus S')$, where $|S''| = \log n$.<br>
**for** every partition $P'$ of $S'$ into $k$ clusters<br>
    Extend $P'$ into a partition $P''$ of $S''$<br>
    Extend $P''$ into a partition $P$ of $S$<br>
    **if** $score(P) < bestScore$<br>
        $bestScore \leftarrow score(P)$<br>
        $bestPartition \leftarrow P$<br>
**return** $bestPartition$

9. Computing this number is an easy (rather than $\mathcal{NP}$-complete) problem, since we are given $P$. To search for the minimum $score$ without $P$, we would need to search over all partitions.

The number of iterations that PCC requires is given by the number of partitions of set $S'$, which is $k^{|S'|} = k^{\log \log n} = (\log n)^{\log_2 k} = (\log n)^c$. The amount of work done in each iteration is $O(n^2)$, resulting in a running time of $O\left(n^2 (\log n)^c\right)$. Since this is too slow for most applications, a more practical heuristic called CAST is often used.




Define the distance between gene $i$ and cluster $C$ as the average distance between $i$ and all genes in the cluster $C$: $d(i, C) = \frac{\sum_{j \in C} d(i,j)}{|C|}$. Given a threshold $\theta$, a gene $i$ is *close* to cluster $C$ if $d(i, C) < \theta$ and *distant* otherwise. The CAST algorithm below clusters set $S$ according to the distance graph $G$ and the threshold $\theta$. CAST iteratively builds the partition $P$ of the set $S$ by finding a cluster $C$ such that no gene $i \notin C$ is close to $C$, and no gene $i \in C$ is distant from $C$. In the beginning of the routine, $P$ is initialized to an empty set.

```
CAST(G, θ)
S ← set of vertices in the distance graph G
P ← ∅
while S ≠ ∅
    v ← vertex of maximal degree in the distance graph G.
    C ← {v}
    while there exists a close gene i ∉ C or distant gene i ∈ C
        Find the nearest close gene i ∉ C and add it to C.
        Find the farthest distant gene i ∈ C and remove it from C.
    Add cluster C to the partition P
    S ← S \ C
    Remove vertices of cluster C from the distance graph G
return P
```

Although CAST is a heuristic with no performance guarantee[10] it performs remarkably well with gene expression data.

## 10.5 Evolutionary Trees

In the past, biologists relied on morphological features, like beak shapes or the presence or absence of fins to construct evolutionary trees. Today biologists rely on DNA sequences for the reconstruction of evolutionary trees. Figure 10.7 represents a DNA-based evolutionary tree of bears and raccoons that helped biologists to decide whether the giant panda belongs to the bear family or the raccoon family. This question is not as obvious as it may at first sound, since bears and raccoons diverged just 35 million years ago and they share many morphological features.

For over a hundred years biologists could not agree on whether the giant panda should be classified in the bear family or in the raccoon family. In 1870 an amateur naturalist and missionary, Père Armand David, returned to Paris from China with the bones of the mysterious creature which he called simply "black and white bear." Biologists examined the bones and concluded that they more closely resembled the bones of a red panda than those of bears. Since red pandas were, beyond doubt, part of the raccoon family, giant pandas were also classified as raccoons (albeit big ones).

Although giant pandas look like bears, they have features that are unusual for bears and typical of raccoons: they do not hibernate in the winter like

10. In fact, CAST may not even converge; see the problems at the end of the chapter.

other bears do, their male genitalia are tiny and backward-pointing (like raccoons' genitalia), and they do not roar like bears but bleat like raccoons. As a result, Edwin Colbert wrote in 1938:

> So the quest has stood for many years with the bear proponents and the raccoon adherents and the middle-of-the-road group advancing their several arguments with the clearest of logic, while in the meantime the giant panda lives serenely in the mountains of Szechuan with never a thought about the zoological controversies he is causing by just being himself.

The giant panda classification was finally resolved in 1985 by Steven O'Brien and colleagues who used DNA sequences and algorithms, rather than behavioral and anatomical features, to resolve the giant panda controversy (fig. 10.7). The final analysis demonstrated that DNA sequences provide an important source of information to test evolutionary hypotheses. O'Brien's study used about 500,000 nucleotides to construct the evolutionary tree of bears and raccoons.

Roughly at the same time that Steven O'Brien resolved the giant panda controversy, Rebecca Cann, Mark Stoneking and Allan Wilson constructed an evolutionary tree of humans and instantly created a new controversy. This tree led to the *Out of Africa* hypothesis, which claims that humans have a common ancestor who lived in Africa 200,000 years ago. This study turned the question of human origins into an algorithmic puzzle.

The tree was constructed from mitochondrial DNA (mtDNA) sequences of people of different races and nationalities.[11] Wilson and his colleagues compared sequences of mtDNA from people representing African, Asian, Australian, Caucasian, and New Guinean ethnic groups and found 133 variants of mtDNA. Next, they constructed the evolutionary tree for these DNA sequences that showed a trunk splitting into two major branches. One branch consisted only of Africans, the other included some modern Africans and some people from everywhere else. They concluded that a population of Africans, the first modern humans, forms the trunk and the first branch of the tree while the second branch represents a subgroup that left Africa and later spread out to the rest of the world. All of the mtDNA, even samples from regions of the world far away from Africa, were strikingly similar. This

11. Unlike the bulk of the genome, mitochondrial DNA is passed solely from a mother to her children without recombining with the father's DNA. Thus it is well-suited for studies of recent human evolution. In addition, it quickly accumulates mutations and thus offers a quick-ticking molecular clock.

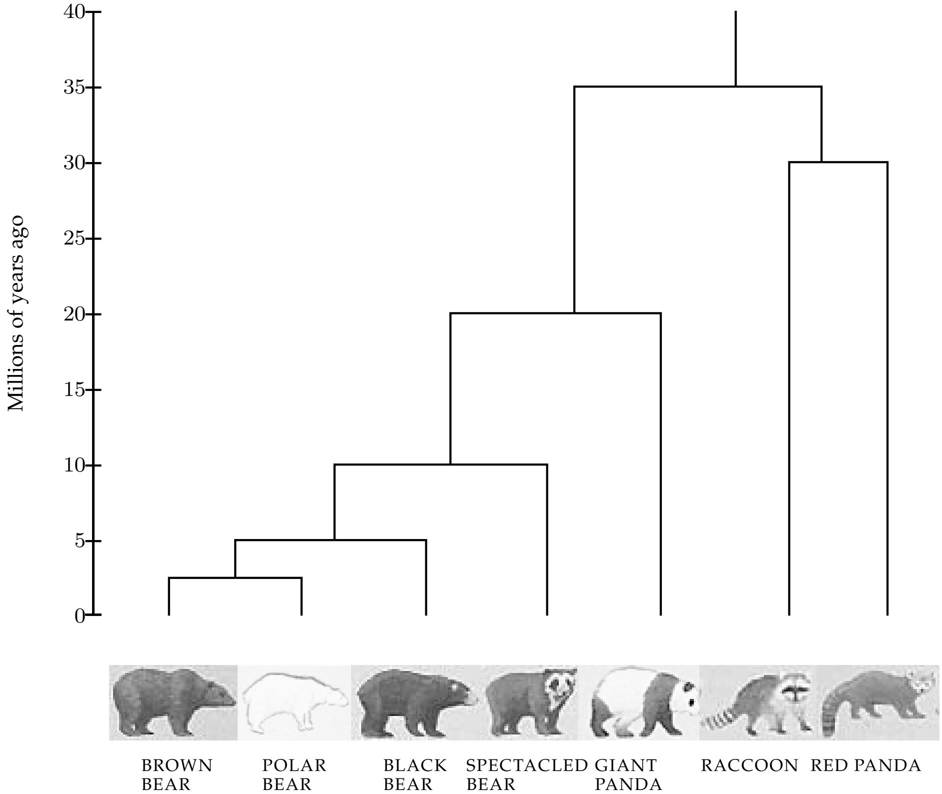



**Figure 10.7** An evolutionary tree showing the divergence of raccoons and bears. Despite their difference in size and shape, these two families are closely related.

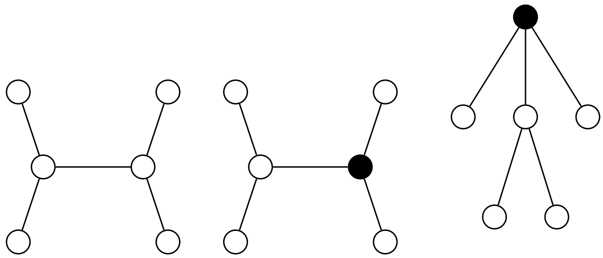
suggested that our species is relatively young. But the African samples had the most mutations, thus implying that the African lineage is the oldest and that all modern humans trace their roots back to Africa. They further estimated that modern man emerged from Africa 200,000 years ago with racial differences arising only 50,000 years ago.

Shortly after Allan Wilson and colleagues constructed the human mtDNA evolutionary tree supporting the Out of Africa hypothesis, Alan Templeton constructed 100 distinct trees that were also consistent with data that provide evidence *against* the African origin hypothesis! This is a cautionary tale suggesting that one should proceed carefully when constructing large evolutionary trees[12] and below we describe some algorithms for evolutionary tree reconstruction.

Biologists use either *unrooted* or *rooted* evolutionary trees;[13] the difference between them is shown in figure 10.8. In a rooted evolutionary tree, the root corresponds to the most ancient ancestor in the tree, and the path from the root to a leaf in the rooted tree is called an *evolutionary path*. Leaves of evolutionary trees correspond to the existing species while internal vertices correspond to hypothetical ancestral species.[14] In the unrooted case, we do not make any assumption about the position of an evolutionary ancestor (root) in the tree. We also remark that rooted trees (defined formally as undirected graphs) can be viewed as directed graphs if one directs the edges of the rooted tree from the root to the leaves.

Biologists often work with *binary weighted* trees where every internal vertex has degree equal to 3 and every edge has an assigned positive weight (sometimes referred to as the *length*). The *weight* of an edge $(v, w)$ may reflect the number of mutations on the evolutionary path from $v$ to $w$ or a time estimate for the evolution of species $v$ into species $w$. We sometimes assume the existence of a *molecular clock* that assigns a time $t(v)$ to every internal vertex $v$ in the tree and a length of $t(w) - t(v)$ to an edge $(v, w)$. Here, time corresponds to the "moment" when the species $v$ produced its descendants; every leaf species corresponds to time 0 and every internal vertex presumably corresponds to some negative time.

12. Following advances in tree reconstruction algorithms, the critique of the Out of Africa hypothesis has diminished in recent years and the consensus today is that this hypothesis is probably correct.
13. We remind the reader that trees are undirected connected graphs that have no cycles. Vertices of degree 1 in the tree are called leaves. All other vertices are called internal vertices.
14. In rare cases like quickly evolving viruses or bacteria, the DNA of ancestral species is available (e.g., as a ten- to twenty-year-old sample stored in refrigerator) thus making sequences of some internal vertices real rather than hypothetical.



(a) Unrooted tree (b) Rooted tree (c) The same rooted tree

**Figure 10.8** The difference between unrooted (a) and rooted (b) trees. These both describe the same tree, but the unrooted tree makes no assumption about the origin of species. Rooted trees are often represented with the root vertex on the top (c), emphasizing that the root corresponds to the ancestral species.

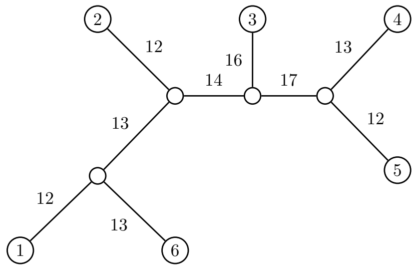



**Figure 10.9** A weighted unrooted tree. The length of the path between any two vertices can be calculated as the sum of the weights of the edges in the path between them. For example, $d(1,5) = 12 + 13 + 14 + 17 + 12 = 68$.

## 10.6 Distance-Based Tree Reconstruction

If we are given a weighted tree $T$ with $n$ leaves, we can compute the length of the path between any two leaves $i$ and $j$, $d_{i,j}(T)$ (fig. 10.9). Evolutionary biologists often face the opposite problem: they measure the $n \times n$ *distance matrix* $(D_{i,j})$, and then must search for a tree $T$ that has $n$ leaves and fits

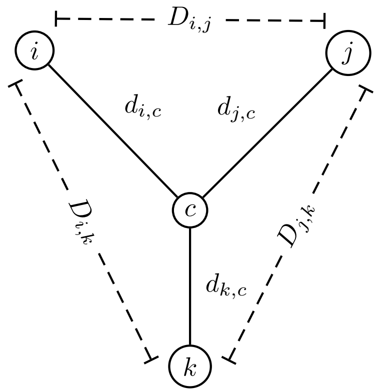



**Figure 10.10** A tree with three leaves.

the data,[15] that is, $d_{i,j}(T) = D_{i,j}$ for every two leaves $i$ and $j$. There are many ways to generate distance matrices: for example, one can sequence a particular gene in $n$ species and define $D_{i,j}$ as the edit distance between this gene in species $i$ and species $j$.

It is not difficult to construct a tree that fits any given $3 \times 3$ (symmetric non-negative) matrix $D$. This binary unrooted tree has four vertices $i, j, k$ as leaves and vertex $c$ as the center. The lengths of each edge in the tree are defined by the following three equations with three variables $d_{i,c}$, $d_{j,c}$, and $d_{k,c}$ (fig. 10.10):

$$d_{i,c} + d_{j,c} = D_{i,j} \qquad d_{i,c} + d_{k,c} = D_{i,k} \qquad d_{j,c} + d_{k,c} = D_{j,k}.$$

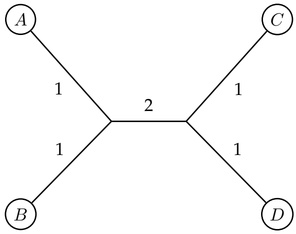
The solution is given by

$$d_{i,c} = \frac{D_{i,j} + D_{i,k} - D_{j,k}}{2} d_{j,c} = \frac{D_{j,i} + D_{j,k} - D_{i,k}}{2} d_{k,c} = \frac{D_{k,i} + D_{k,j} - D_{i,j}}{2}.$$

An unrooted binary tree with $n$ leaves has $2n - 3$ edges, so fitting a *given* tree to an $n \times n$ distance matrix $D$ leads to solving a system of $\binom{n}{2}$ equations with $2n - 3$ variables. For $n = 4$ this amounts to solving six equations with only five variables. Of course, it is not always possible to solve this system,

15. We assume that all matrices in this chapter are symmetric, that is, that they satisfy the conditions $D_{i,j} = D_{j,i}$ and $D_{i,j} \geq 0$ for all $i$ and $j$. We also assume that the distance matrices satisfy the triangle inequality, that is, $D_{i,j} + D_{j,k} \geq D_{i,k}$ for all $i$, $j$, and $k$.

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 4 | 4 |
| B | 2 | 0 | 4 | 4 |
| C | 4 | 4 | 0 | 2 |
| D | 4 | 4 | 2 | 0 |




(a) Additive matrix and the corresponding tree

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 2 | 2 |
| B | 2 | 0 | 3 | 2 |
| C | 2 | 3 | 0 | 2 |
| D | 2 | 2 | 2 | 0 |

?

(b) Non-additive matrix

**Figure 10.11** Additive and nonadditive matrices.

making it hard or impossible to construct a tree from $D$. A matrix $(D_{i,j})$ is called *additive* if there exists a tree $T$ with $d_{i,j}(T) = D_{i,j}$, and *nonadditive* otherwise (fig. 10.11).

---

**Distance-Based Phylogeny Problem**:
*Reconstruct an evolutionary tree from a distance matrix.*

**Input:** An $n \times n$ distance matrix $(D_{i,j})$.

**Output:** A weighted unrooted tree $T$ with $n$ leaves fitting $D$, that is, a tree such that $d_{i,j}(T) = D_{i,j}$ for all $1 \le i < j \le n$ if $(D_{i,j})$ is additive.

---

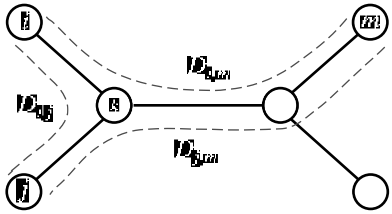


**Figure 10.12** If $i$ and $j$ are neighboring leaves and $k$ is their parent, then $D_{k,m} = \frac{D_{i,m}+D_{j,m}-D_{i,j}}{2}$ for every other vertex $m$ in the tree.

The Distance-Based Phylogeny problem may not have a solution, but if it does—that is, if $D$ is additive—there exists a simple algorithm to solve it. We emphasize the fact that we are somehow given the matrix of evolutionary distances between each pair of species, and we are searching for both the shape of the tree that fits this distance matrix and the weights for each edge in the tree.

## 10.7 Reconstructing Trees from Additive Matrices

A "simple" way to solve the Distance-Based Phylogeny problem for additive trees[16] is to find a pair of *neighboring* leaves, that is, leaves that have the same parent vertex.[17] Figure 10.12 illustrates that for a pair of neighboring leaves $i$ and $j$ and their parent vertex $k$, the following equality holds for every other leaf $m$ in the tree:

$$D_{k,m} = \frac{D_{i,m} + D_{j,m} - D_{i,j}}{2}$$

Therefore, as soon as a pair of neighboring leaves $i$ and $j$ is found, one can remove the corresponding rows and columns $i$ and $j$ from the distance matrix and add a new row and column corresponding to their parent $k$. Since the distance matrix is additive, the distances from $k$ to other leaves are recomputed as $D_{k,m} = \frac{D_{i,m}+D_{j,m}-D_{i,j}}{2}$. This transformation leads to a simple algorithm for the Distance-Based Phylogeny problem that finds a pair of neighboring leaves and reduces the size of the tree at every step.

One problem with the described approach is that it is not very easy to find neighboring leaves! One might be tempted to think that a pair of closest

16. To be more precise, we mean an "additive matrix," rather than an "additive tree"; the term "additive" applies to matrices. We use the term "additive trees" only because it dominates the

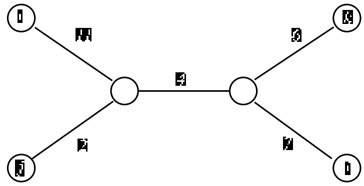


**Figure 10.13** The two closest leaves ($j$ and $k$) are not neighbors in this tree.

leaves (i.e., the leaves $i$ and $j$ with minimum $D_{i,j}$) would represent a pair of neighboring leaves, but a glance at figure 10.13 will show that this is not true. Since finding neighboring leaves using only the distance matrix is a nontrivial problem, we postpone exploring this until the next section and turn to another approach.

Figure 10.14 illustrates the process of shortening *all* "hanging" edges of a tree $T$, that is, edges leading to leaves. If we reduce the length of every hanging edge by the same small amount $\delta$, then the distance matrix of the resulting tree is $(d_{i,j} - 2\delta)$ since the distance between any two leaves is reduced by $2\delta$. Sooner or later this process will lead to "collapsing" one of the leaves when the length of the corresponding hanging edge becomes equal to $0$ (when $\delta$ is equal to the length of the shortest hanging edge). At this point, the original tree $T = T_n$ with $n$ leaves will be transformed into a tree $T_{n-1}$ with $n - 1$ or fewer leaves.

Although the distance matrix $D$ does not explicitly contain information about $\delta$, it is easy to derive both $\delta$ and the location of the collapsed leaf in $T_{n-1}$ by the method described below. Thus, one can perform a series of tree transformations $T_n \rightarrow T_{n-1} \rightarrow \ldots \rightarrow T_3 \rightarrow T_2$, then construct the tree $T_2$ (which is easy, since it consists of only a single edge), and then perform a series of reverse transformations $T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_{n-1} \rightarrow T_n$ recovering information about the collapsed edges at every step (fig. 10.14)

A triple of distinct elements $1 \leq i, j, k \leq n$ is called *degenerate* if $D_{i,j} + D_{j,k} = D_{i,k}$, which is essentially just an indication that $j$ is located on the path from $i$ to $k$ in the tree. If $D$ is additive, $D_{i,j} + D_{j,k} \geq D_{i,k}$ for every triple $i, j, k$. We call the entire matrix $D$ degenerate if it has at least one degenerate triple. If $(i, j, k)$ is a degenerate triple, and some tree $T$ fits matrix $D$, then the

literature on evolutionary tree reconstruction.

17. Showing that every binary tree has neighboring leaves is left as a problem at the end of this chapter.

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 4 | 10 | 9 |
| B | 4 | 0 | 8 | 7 |
| C | 10 | 8 | 0 | 9 |
| D | 9 | 7 | 9 | 0 |

$\downarrow \delta = 1$

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 2 | 8 | 7 |
| B | 2 | 0 | 6 | 5 |
| C | 8 | 6 | 0 | 7 |
| D | 7 | 5 | 7 | 0 |

$i \leftarrow A$
$j \leftarrow B$
$k \leftarrow C$

$\downarrow$

| | A | C | D |
|---|---|---|---|
| A | 0 | 8 | 7 |
| C | 8 | 0 | 7 |
| D | 7 | 7 | 0 |

$\downarrow \delta = 3$

| | A | C | D |
|---|---|---|---|
| A | 0 | 2 | 1 |
| C | 2 | 0 | 1 |
| D | 1 | 1 | 0 |

$i \leftarrow A$
$j \leftarrow D$
$k \leftarrow C$

$\downarrow$

| | A | C |
|---|---|---|
| A | 0 | 2 |
| C | 2 | 0 |

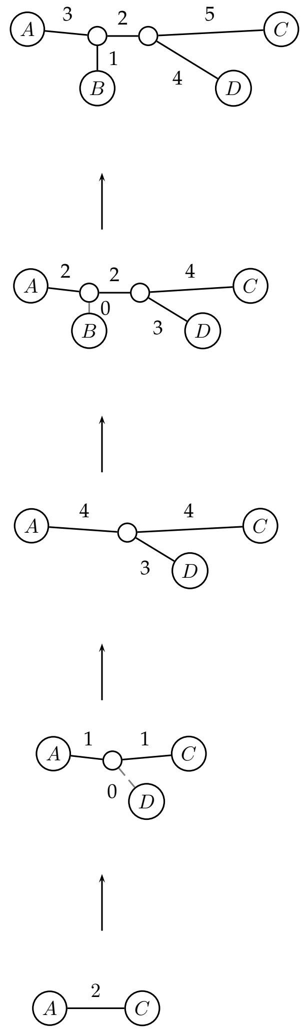


**Figure 10.14** The iterative process of shortening the hanging edges of a tree.

vertex $j$ should lie somewhere on the path between $i$ and $k$ in $T$.[18] Another way to state this is that $j$ is attached to this path by an edge of weight 0, and the attachment point for $j$ is located at distance $D_{i,j}$ from vertex $i$. Therefore, if an $n \times n$ additive matrix $D$ has a degenerate triple, then it will be reduced to an $(n-1) \times (n-1)$ additive matrix by simply excluding vertex $j$ from consideration; the position of $j$ will be recovered during the reverse transformations. If the matrix $D$ does not have a degenerate triple, one can start reducing the values of all elements in $D$ by the same amount $2\delta$ until the point at which the distance matrix becomes degenerate for the first time (i.e., $\delta$ is the minimum value for which $(D_{i,j} - 2\delta)$ has a degenerate triple for some $i$ and $j$). Determining how to calculate the minimum value of $\delta$ (called the *trimming parameter*) is left as a problem at the end of this chapter. Though you do not have the tree $T$, this operation corresponds to shortening all of the hanging edges in $T$ by $\delta$ until one of the leaves ends up on the evolutionary path between two other leaves for the first time. This intuition motivates the following recursive algorithm for finding the tree that fits the data.

AdditivePhylogeny($D$)
**if** $D$ is a $2 \times 2$ matrix
    $T \leftarrow$ the tree consisting of a single edge of length $D_{1,2}$.
    **return** $T$
**if** $D$ is non-degenerate
    $\delta \leftarrow$ trimming parameter of matrix $D$
    **for** all $1 \leq i \neq j \leq n$
        $D_{i,j} \leftarrow D_{i,j} - 2\delta$
**else**
    $\delta \leftarrow 0$
Find a triple $i, j, k$ in $D$ such that $D_{ij} + D_{jk} = D_{ik}$
$x \leftarrow D_{i,j}$
Remove $j$th row and $j$th column from $D$.
$T \leftarrow$ AdditivePhylogeny($D$)
Add a new vertex $v$ to $T$ at distance $x$ from $i$ to $k$
Add $j$ back to $T$ by creating an edge $(v, j)$ of length 0
**for** every leaf $l$ in $T$
    **if** distance from $l$ to $v$ in the tree $T$ does not equal $D_{l,j}$
        **output** "Matrix $D$ is not additive"
        **return**
Extend hanging edges leading to all leaves by $\delta$
**return** $T$

18. To be more precise, vertex $j$ partitions the path from $i$ to $k$ into paths of length $D_{i,j}$ and $D_{j,k}$.

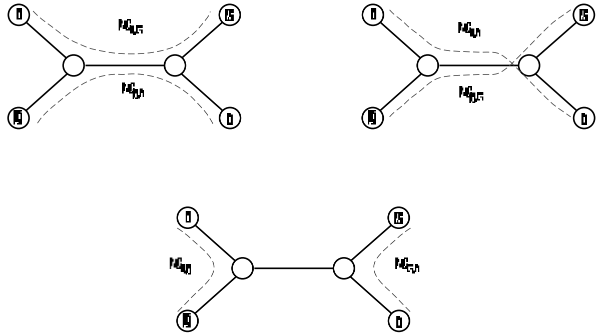


**Figure 10.15** Representing three sums in a tree with $4$ vertices.

The ADDITIVEPHYLOGENY algorithm above provides a way to check if the matrix $D$ is additive. While this algorithm is intuitive and simple, it is not the most efficient way to construct additive trees. Another way to check additivity is by using the following "four-point condition". Let $1 \leq i, j, k, l \leq n$ be four distinct indices. Compute 3 sums: $D_{i,j} + D_{k,l}$, $D_{i,k} + D_{j,l}$, and $D_{i,l} + D_{j,k}$. If $D$ is an additive matrix then these three sums can be represented by a tree with four leaves (fig. 10.15). Moreover, two of these sums represent the same number (the sum of lengths of all edges in the tree plus the length of the middle edge) while the third sum represents another smaller number (the sum of lengths of all edges in the tree minus the length of the middle edge). We say that elements $1 \leq i, j, k, l \leq n$ satisfy the *four-point condition* if two of the sums $D_{i,j} + D_{k,l}$, $D_{i,k} + D_{j,l}$, and $D_{i,l} + D_{j,k}$ are the same, and the third one is smaller than these two.

**Theorem 10.1** *An $n \times n$ matrix $D$ is additive if and only if the four point condition holds for every 4 distinct elements $1 \leq i, j, k, l \leq n$.*

If the distance matrix $D$ is not additive, one might want instead to find a tree that approximates $D$ using the sum of squared errors $\sum_{i,j}(d_{i,j}(T) - D_{i,j})^2$ as a measure of the quality of the approximation. This leads to the ($\mathcal{NP}$-hard) Least Squares Distance-Based Phylogeny problem:

**Least Squares Distance-Based Phylogeny Problem**:
*Given a distance matrix, find the evolutionary tree that minimizes squared error.*

**Input:** An $n \times n$ distance matrix $(D_{i,j})$

**Output:** A weighted tree $T$ with $n$ leaves minimizing $\sum_{i,j}(d_{i,j}(T) - D_{i,j})^2$ over all weighted trees with $n$ leaves.

## 10.8 Evolutionary Trees and Hierarchical Clustering

Biologists often use variants of hierarchical clustering to construct evolutionary trees. UPGMA (*Unweighted Pair Group Method with Arithmetic Mean*) is a particularly simple clustering algorithm. The UPGMA algorithm is a variant of HIERARCHICALCLUSTERING that uses a different approach to compute the distance between clusters, and assigns *heights* to vertices of the constructed tree. Thus, the length of an edge $(u, v)$ is defined to be the difference in heights of the vertices $v$ and $u$. The height plays the role of the molecular clock, and allows one to "date" the divergence point for every vertex in the evolutionary tree.

Given clusters $C_1$ and $C_2$, UPGMA defines the distance between them to be the average pairwise distance: $D(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{i \in C_1} \sum_{j \in C_2} D(i, j)$. At heart, UPGMA is simply another hierarchical clustering algorithm that "dates" vertices of the constructed tree.

UPGMA$(D, n)$
Form $n$ clusters, each with a single element
Construct a graph $T$ by assigning an isolated vertex to each cluster
Assign height $h(v) = 0$ to every vertex $v$ in this graph
**while** there is more than one cluster
    Find the two closest clusters $C_1$ and $C_2$
    Merge $C_1$ and $C_2$ into a new cluster $C$ with $|C_1| + |C_2|$ elements
    **for** every cluster $C^* \neq C$
        $D(C, C^*) = \frac{1}{|C| \cdot |C^*|} \sum_{i \in C} \sum_{j \in C^*} D(i, j)$
    Add a new vertex $C$ to $T$ and connect to vertices $C_1$ and $C_2$
    $h(C) \leftarrow \frac{D(C_1, C_2)}{2}$
    Assign length $h(C) - h(C_1)$ to the edge $(C_1, C)$
    Assign length $h(C) - h(C_2)$ to the edge $(C_2, C)$
    Remove rows and columns of $D$ corresponding to $C_1$ and $C_2$
    Add a row and column to $D$ for the new cluster $C$
**return** $T$

UPGMA produces a special type of rooted tree[19] that is known as *ultrametric*. In ultrametric trees the distance from the root to any leaf is the same.

We can now return to the "neighboring leaves" idea that we developed and then abandoned in the previous section. In 1987 Naruya Saitou and Masatoshi Nei developed an ingenious *neighbor joining* algorithm for phylogenetic tree reconstruction. In the case of additive trees, the neighbor joining algorithm somehow magically finds pairs of neighboring leaves and proceeds by substituting such pairs with the leaves' parent. However, neighbor joining works well not only for additive distance matrices but for many others as well: it does not assume the existence of a molecular clock and ensures that the clusters that are merged in the course of tree reconstruction are not only close to each other (as in UPGMA) but also are far apart from the rest.

For a cluster $C$, define $u(C) = \frac{1}{\text{number of clusters}-2} \sum_{\text{all clusters } C'} D(C, C')$ as a measure of the separation of $C$ from other clusters.[20] To choose which two clusters to merge, we look for the clusters $C_1$ and $C_2$ that are simultaneously close to each other and far from others. One may try to merge clusters that simultaneously minimize $D(C_1, C_2)$ and maximize $u(C_1) + u(C_2)$. However, it is unlikely that a pair of clusters $C_1$ and $C_2$ that simultaneously minimize $D(C_1, C_2)$ and maximize $u(C_1) + u(C_2)$ exists. As an alternative, one opts to minimize $D(C_1, C_2) - u(C_1) - u(C_2)$. This approach is used in the NEIGHBORJOINING algorithm below.

NEIGHBORJOINING$(D, n)$
Form $n$ clusters, each with a single element
Construct a graph $T$ by assigning an isolated vertex to each cluster
**while** there is more than one cluster
    Find clusters $C_1$ and $C_2$ minimizing $D(C_1, C_2) - u(C_1) - u(C_2)$
    Merge $C_1$ and $C_2$ into a new cluster $C$ with $|C_1| + |C_2|$ elements
    Compute $D(C, C^*) = \frac{D(C_1,C) + D(C_2,C)}{2}$ to every other cluster $C^*$
    Add a new vertex $C$ to $T$ and connect it to vertices $C_1$ and $C_2$
    Assign length $\frac{1}{2}D(C_1, C_2) + \frac{1}{2}(u(C_1) - u(C_2))$ to the edge $(C_1, C)$
    Assign length $\frac{1}{2}D(C_1, C_2) + \frac{1}{2}(u(C_2) - u(C_1))$ to the edge $(C_2, C)$
    Remove rows and columns of $D$ corresponding to $C_1$ and $C_2$
    Add a row and column to $D$ for the new cluster $C$
**return** $T$

19. Here, the root corresponds to the cluster created last.
20. The explanation of that mysterious term "number of clusters $-$ 2" in this formula is beyond the scope of this book.

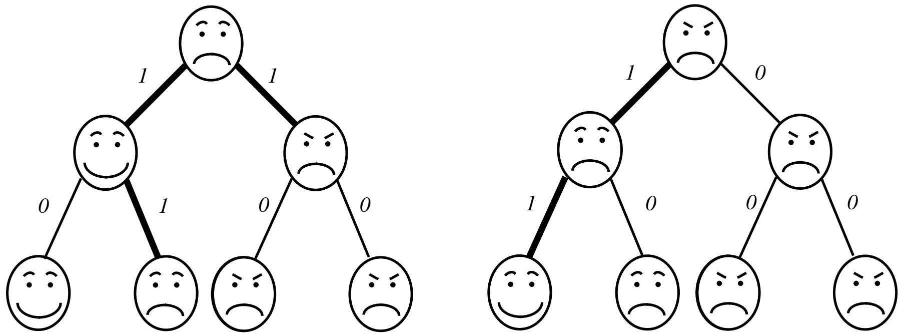
## 10.9 Character-Based Tree Reconstruction

Evolutionary tree reconstruction often starts by sequencing a particular gene in each of $n$ species. After aligning these genes, biologists end up with an $n \times m$ alignment matrix ($n$ species, $m$ nucleotides in each) that can be further transformed into an $n \times n$ distance matrix. Although the distance matrix could be analyzed by distance-based tree reconstruction algorithms, a certain amount of information gets lost in the transformation of the alignment matrix into the distance matrix, rendering the reverse transformation of distance matrix back into the alignment matrix impossible. A better technique is to use the alignment matrix directly for evolutionary tree reconstruction. *Character-based* tree reconstruction algorithms assume that the input data are described by an $n \times m$ matrix (perhaps an alignment matrix), where $n$ is the number of species and $m$ is the number of *characters*. Every row in the matrix describes an existing species and the goal is to construct a tree whose leaves correspond to the $n$ existing species and whose internal vertices correspond to ancestral species. Each internal vertex in the tree is labeled with a character string that describes, for example, the hypothetical number of legs in that ancestral species. We want to determine what character strings at internal nodes would best explain the character strings for the $n$ observed species.

The use of the word "character" to describe an attribute of a species is potentially confusing, since we often use the word to refer to letters from an alphabet. We are not at liberty to change the terminology that biologists have been using for at least a century, so for the next section we will refer to nucleotides as *states* of a character. Another possible character might be "number of legs," which is not very informative for mammalian evolutionary studies, but could be somewhat informative for insect evolutionary studies.

An intuitive score for a character-based evolutionary tree is the total number of mutations required to explain all of the observed character sequences. The *parsimony* approach attempts to minimize this score, and follows the philosophy of Occam's razor: find the simplest explanation of the data (see figure 10.16).[21]

21. Occam was one of the most influential philosophers of the fourteenth century. Strong opposition to his ideas from theology professors prevented him from obtaining his masters degree (let alone his doctorate). Occam's razor states that "plurality should not be assumed without necessity," and is usually paraphrased as "keep it simple, stupid." Occam used this principle to eliminate many pseudoexplanatory theological arguments. Though the parsimony principle is attributed to Occam, sixteen centuries earlier Aristotle wrote simply that *Nature operates in the shortest way possible*.




(a) *Parsimony Score=3* (b) *Parsimony Score=2*

**Figure 10.16** If we label a tree's leaves with characters (in this case, eyebrows and mouth, each with two states), and choose labels for each internal vertex, we implicitly create a *parsimony* score for the tree. By changing the labels in (a) we are able to create a tree with a better parsimony score in (b).

Given a tree $T$ with every vertex labeled by an $m$-long string of characters, one can set the length of an edge $(v, w)$ to the Hamming distance $d_H(v, w)$ between the character strings for $v$ and $w$. The *parsimony score* of a tree $T$ is simply the sum of lengths of its edges $\sum_{All\ edges\ (v,w)\ of\ the\ tree} d_H(v, w)$. In reality, the strings of characters assigned to internal vertices are unknown and the problem is to find strings that minimize the parsimony score.

Two particular incarnations of character-based tree reconstruction are the Small Parsimony problem and the Large Parsimony problem. The Small Parsimony problem assumes that the tree is given but the labels of its internal vertices are unknown, while the vastly more difficult Large Parsimony problem assumes that neither the tree structure nor the labels of its internal vertices are known.

## 10.10 Small Parsimony Problem

---

**Small Parsimony Problem**:
*Find the most parsimonious labeling of the internal vertices in an evolutionary tree.*

> **Input:** Tree $T$ with each leaf labeled by an $m$-character string.
>
> **Output:** Labeling of internal vertices of the tree $T$ minimizing the parsimony score.

---

An attentive reader should immediately notice that, because the characters in the string are independent, the Small Parsimony problem can be solved independently for each character. Therefore, to devise an algorithm, we can assume that every leaf is labeled by a single character rather than by a string of $m$ characters.

As we have seen in previous chapters, sometimes solving a more general—and seemingly more difficult—problem may reveal the solution to the more specific one. In the case of the Small Parsimony Problem we will first solve the more general *Weighted Small Parsimony* problem, which generalizes the notion of parsimony by introducing a scoring matrix. The length of an edge connecting vertices $v$ and $w$ in the Small Parsimony problem is defined as the Hamming distance, $d_H(v, w)$, between the character strings for $v$ and $w$. In the case when every leaf is labeled by a single character in a $k$-letter alphabet, $d_H(v, w) = 0$ if the characters corresponding to $v$ and $w$ are the same, and $d_H(v, w) = 1$ otherwise. One can view such a scoring scheme as a $k \times k$ scoring matrix $(\delta_{i,j})$ with diagonal elements equal to 0 and all other elements equal to 1. The Weighted Small Parsimony problem simply assumes that the scoring matrix $(\delta_{i,j})$ is an arbitrary $k \times k$ matrix and minimizes the weighted parsimony score $\sum_{\text{all edges } (v,\, w) \text{ in the tree}} \delta_{v,w}$.

---

**Weighted Small Parsimony Problem**:
*Find the minimal weighted parsimony score labeling of the internal vertices in an evolutionary tree.*

> **Input:** Tree $T$ with each leaf labeled by elements of a $k$-letter alphabet and a $k \times k$ scoring matrix $(\delta_{ij})$.
>
> **Output:** Labeling of internal vertices of the tree $T$ minimizing the weighted parsimony score.

---

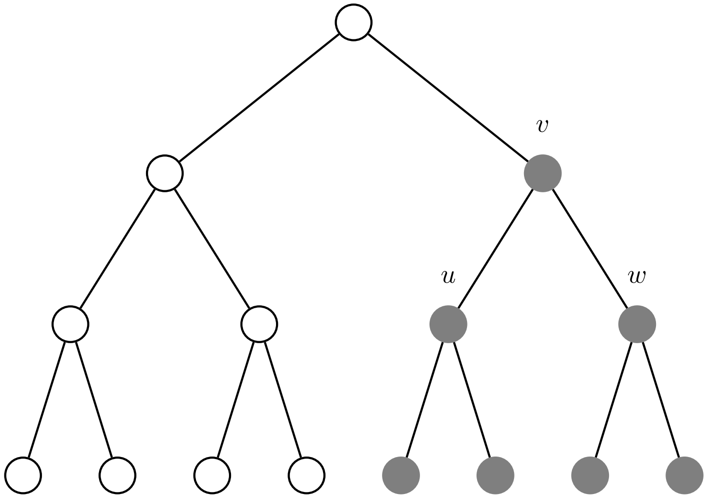



**Figure 10.17** A subtree of a larger tree. The shaded vertices form a tree rooted at the topmost shaded node.

In 1975 David Sankoff came up with the following dynamic programming algorithm for the Weighted Small Parsimony problem. As usual in dynamic programming, the Weighted Small Parsimony problem for $T$ is reduced to solving the Weighted Small Parsimony Problem for smaller subtrees of $T$. As we mentioned earlier, a rooted tree can be viewed as a directed tree with all of its edges directed away from the root toward the leaves. Every vertex $v$ in the tree $T$ defines a *subtree* formed by the vertices beneath $v$ (fig. 10.17), which are all of the vertices that can be reached from $v$. Let $s_t(v)$ be the minimum parsimony score of the subtree of $v$ under the assumption that vertex $v$ has character $t$. For an internal vertex $v$ with children $u$ and $w$, the score $s_t(v)$ can be computed by analyzing $k$ scores $s_i(u)$ and $k$ scores $s_i(w)$ for $1 \leq i \leq k$ (below, $i$ and $j$ are characters):

$$s_t(v) = \min_i \{s_i(u) + \delta_{i,t}\} + \min_j \{s_j(w) + \delta_{j,t}\}$$

The initial conditions simply amount to an assignment of the scores $s_t(v)$ at the leaves according to the rule: $s_t(v) = 0$ if $v$ is labeled by letter $t$ and $s_t(v) = \infty$ otherwise. The minimum weighted parsimony score is given by the smallest score at the root, $s_t(r)$ (fig. 10.18). Given the computed values

$s_t(v)$ at all of the vertices in the tree, one can reconstruct an optimal assignment of labels using a backtracking approach that is similar to that used in chapter 6. The running time of the algorithm is $O(nk)$.

In 1971, even before David Sankoff solved the Weighted Small Parsimony problem, Walter Fitch derived a solution of the (unweighted) Small Parsimony problem. The Fitch algorithm below is essentially dynamic programming in disguise. The algorithm assigns a set of letters $S_v$ to every vertex in the tree in the following manner. For each leaf $v$, $S_v$ consists of single letter that is the label of this leaf. For any internal vertex $v$ with children $u$ and $w$, $S_v$ is computed from the sets $S_u$ and $S_w$ according to the following rule:

$$S_v = \begin{cases} S_u \cap S_w, & \text{if } S_u \text{ and } S_w \text{ overlap} \\ S_u \cup S_w, & \text{otherwise} \end{cases}$$

To compute $S_v$, we traverse the tree in *post-order* as in figure 10.19, starting from the leaves and working toward the root. After computing $S_v$ for all vertices in the tree, we need to decide on how to assign letters to the internal vertices of the tree. This time we traverse the tree using *preorder* traversal from the root toward the leaves. We can assign root $r$ any letter from $S_r$. To assign a letter to an internal vertex $v$ we check if the (already assigned) label of its parent belongs to $S_v$. If yes, we choose the same label for $v$; otherwise we label $v$ by an arbitrary letter from $S_v$ (fig. 10.20). The running time of this algorithm is also $O(nk)$.

At first glance, Fitch's labeling procedure and Sankoff's dynamic programming algorithm appear to have little in common. Even though Fitch probably did not know about application of dynamic programming for evolutionary tree reconstruction in 1971, the two algorithms are almost identical. To reveal the similarity between these two algorithms let us return to Sankoff's recurrence. We say that character $t$ is optimal for vertex $v$ if it yields the smallest score, that is, if $s_t(v) = \min_{1 \le i \le k} s_i(v)$. The set of optimal letters for a vertex $v$ forms a set $S(v)$. If $u$ and $w$ are children of $v$ and if $S(u)$ and $S(w)$ overlap, then it is easy to see that $S(v) = S(u) \bigcap S(w)$. If $S(u)$ and $S(w)$ do not overlap, then it is easy to see that $S(v) = S(u) \bigcup S(w)$. Fitch's algorithm uses exactly the same recurrence, thus revealing that these two approaches are algorithmic twins.

| $\delta$ | A | T | G | C |
|---|---|---|---|---|
| A | 0 | 3 | 4 | 9 |
| T | 3 | 0 | 2 | 4 |
| G | 4 | 2 | 0 | 4 |
| C | 9 | 4 | 4 | 0 |

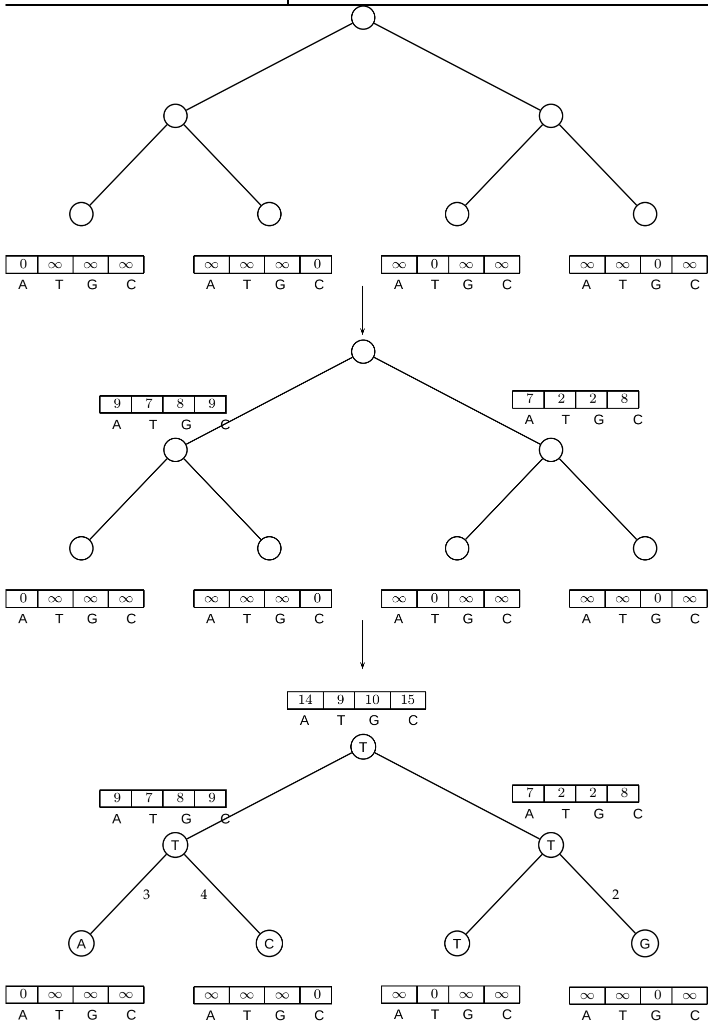



**Figure 10.18** An illustration of Sankoff's algorithm. The leaves of the tree are labeled by A, C, T, G in order. The minimum weighted parsimony score is given by $s_{\mathsf{T}}(root) = 0 + 0 + 3 + 4 + 0 + 2 = 9$.

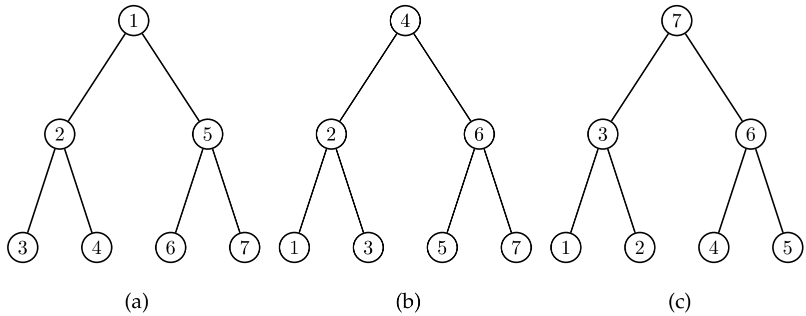



**Figure 10.19** Three methods of traversing a tree. (a) Pre-order: SELF, LEFT, RIGHT. (b) In-order: LEFT, SELF, RIGHT. (c) Post-order: LEFT, RIGHT, SELF.

## 10.11 Large Parsimony Problem

---

**Large Parsimony Problem**:
*Find a tree with $n$ leaves having the minimal parsimony score.*

**Input:** An $n \times m$ matrix $M$ describing $n$ species, each represented by an $m$-character string.

**Output:** A tree $T$ with $n$ leaves labeled by the $n$ rows of matrix $M$, and a labeling of the internal vertices of this tree such that the parsimony score is minimized over all possible trees and over all possible labelings of internal vertices.

---

Not surprisingly, the Large Parsimony problem is $\mathcal{NP}$-complete. In the case $n$ is small, one can explore all tree topologies with $n$ leaves, solve the Small Parsimony problem for each topology, and select the best one. However, the number of topologies grows very fast with respect to $n$, so biologists often use local search heuristics (e.g., greedy algorithms) to navigate in the space of all topologies. *Nearest neighbor interchange* is a local search heuristic that defines neighbors in the space of all trees.[22] Every *internal* edge in a tree defines four subtrees $A, B, C$, and $D$ (fig. 10.21) that can be combined into a

22. "Nearest neighbor" has nothing to do with the two closest leaves in the tree.

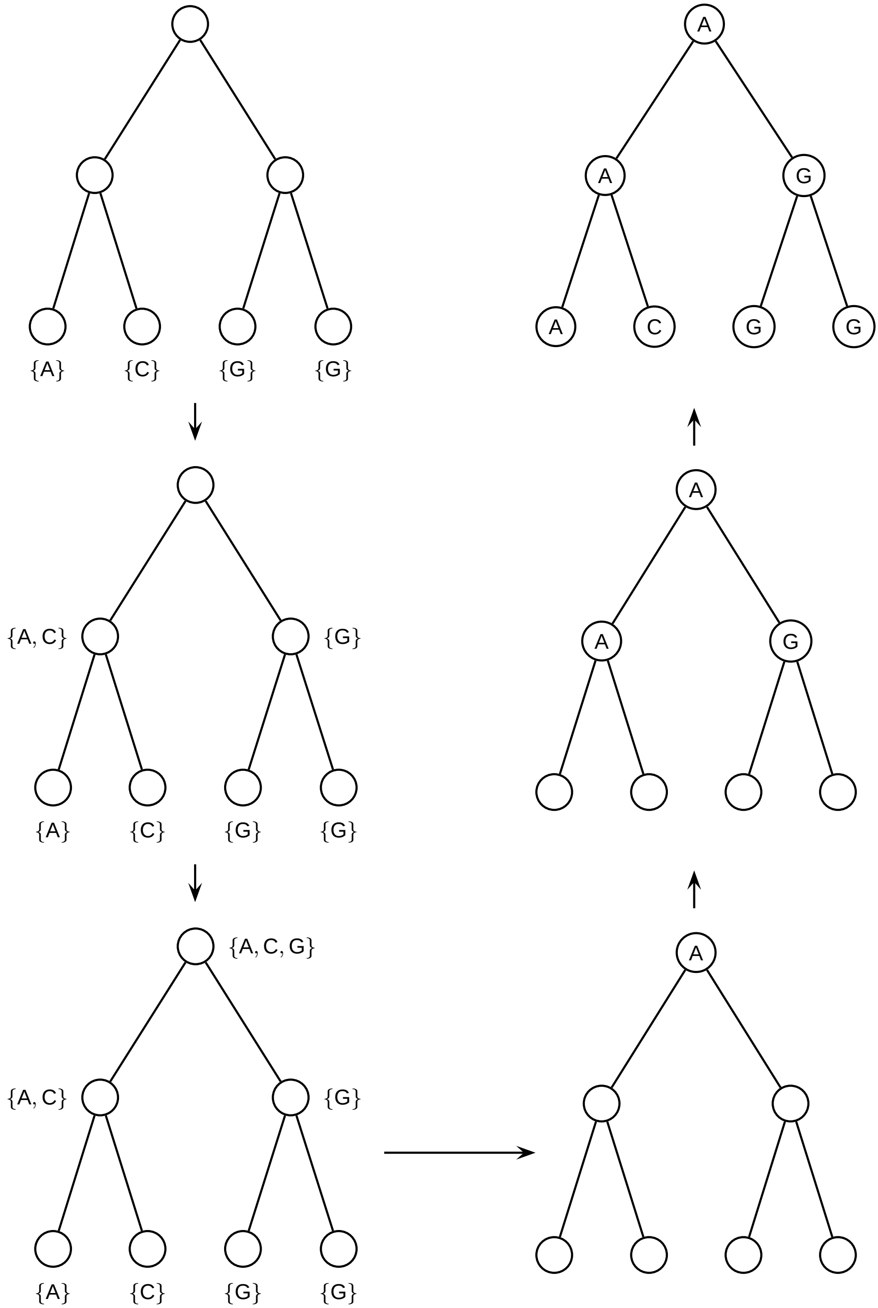



**Figure 10.20** An illustration of Fitch's algorithm.

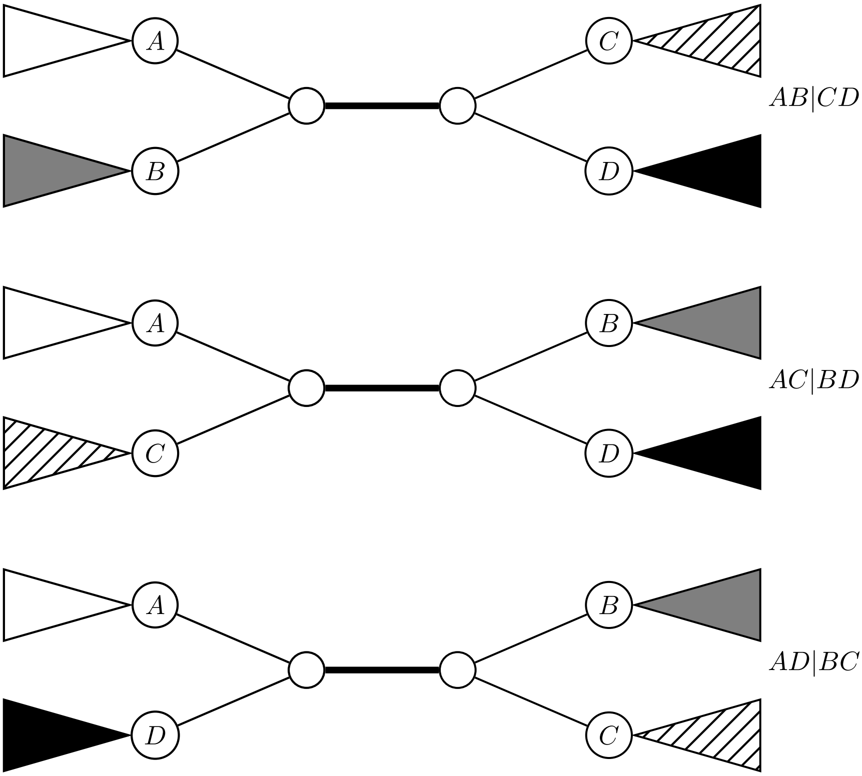
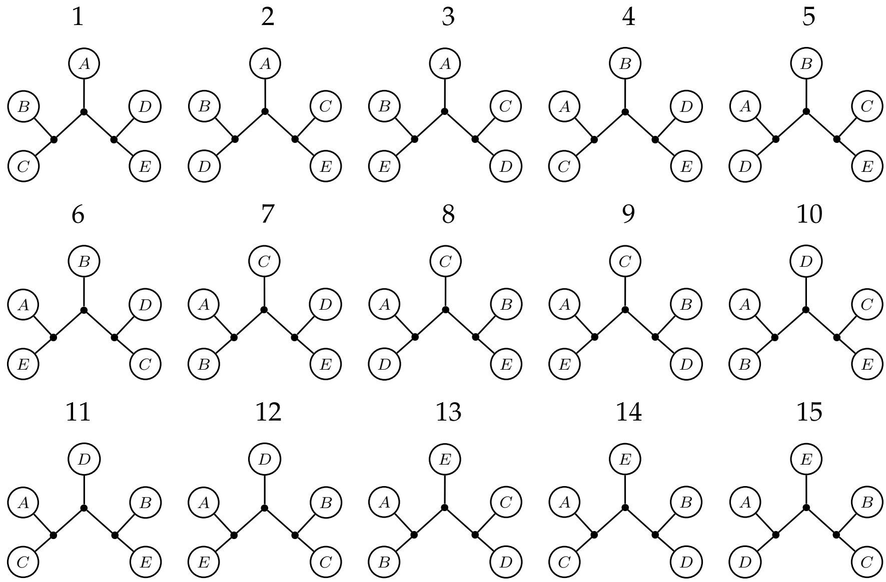
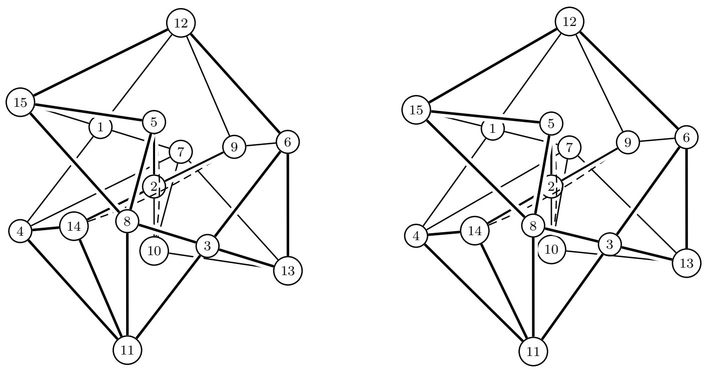



**Figure 10.21** Three ways of combining the four subtrees defined by an internal edge.

tree in three different ways that we denote $AB|CD$, $AC|BD$, and $AD|BC$. These three trees are called *neighbors* under the nearest neighbor interchange transformation. Figure 10.22 shows all trees with five leaves and connects two trees if they are neighbors. Figure 10.23 shows two nearest neighbor interchanges that transform one tree into another. A greedy approach to the Large Parsimony problem is to start from an arbitrary tree and to move (by nearest neighbor interchange) from one tree to another if such a move provides the best improvement in the parsimony score among all neighbors of the tree $T$.




(a) All 5-leaf binary trees




(b) Stereo projection of graph of trees

**Figure 10.22** (a) All unrooted binary trees with five leaves. (b) These can also be considered to be vertices in a graph; two vertices are connected if and only if their respective trees are interchangeable by a single nearest neighbor interchange operation. Shown is a three dimensional view of the graph as a stereo representation.

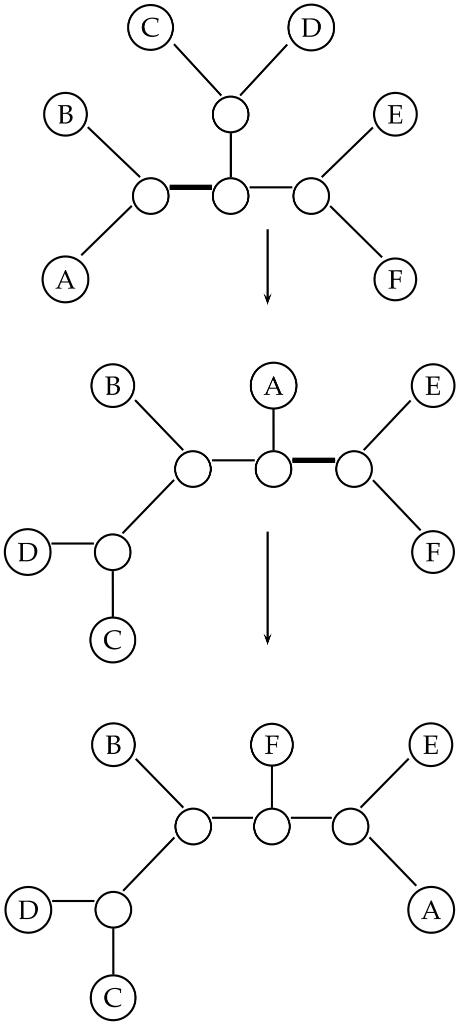



**Figure 10.23** Two trees that are two nearest neighbor interchanges apart.

## 10.12 Notes

The literature on clustering can be traced back to the nineteenth century. The $k$-Means clustering algorithm was introduced by Stuart Lloyd in 1957 (68), popularized by MacQueen in 1965 (70), and has since inspired dozens of variants. Applications of hierarchical clustering for gene expression analyses were pioneered by Michael Eisen, Paul Spellman, Patrick Brown and David Botstein in 1998 (33). The PCC and CAST algorithms were developed by Amir Ben-Dor, Ron Shamir, and Zohar Yakhini in 1999 (11).

The molecular solution of the giant panda riddle was proposed by Stephen O'Brien and colleagues in 1985 (81) and described in more details in O'Brien's book (80). The Out of Africa hypothesis was proposed by Rebecca Cann, Mark Stoneking, and Allan Wilson in 1987. Even today it continues to be debated by Alan Templeton and others (103).

The simple and intuitive UPGMA hierarchical clustering approach was developed in 1958 by Charles Michener and Robert Sokal (98). The neighbor joining algorithm was developed by Naruya Saitou and Masatoshi Nei in 1987 (90).

The algorithm to solve the Small Parsimony problem was developed by Walter Fitch in 1971 (37), four years before David Sankoff developed a dynamic programming algorithm for the more general Weighted Small Parsimony problem (92). The four-point condition was first formulated in 1965 (113), promptly forgotten, and rediscovered by Peter Buneman six years later (19). The nearest neighbor interchange approach to exploring trees was proposed in 1971 by David Robinson (88).

**Ron Shamir** (born 1953 in Jerusalem) is a professor at the School of Computer Science at Tel Aviv University. He holds an undergraduate degree in mathematics and physics from Hebrew University, Jerusalem and a PhD in operations research from the University of California at Berkeley. He was a pioneer in the application of sophisticated graph-theoretical techniques in bioinformatics. Shamir's interests have always revolved around the design and analysis of algorithms and his PhD dissertation was on studies of linear programming algorithms. Later he developed a keen interest in graph algorithms that eventually brought him (in a roundabout way) to computational biology. Back in 1990 Shamir was collaborating with Martin Golumbic on problems related to temporal reasoning. In these problems, one has a set of events, each modeled by an unknown interval on the timeline, and a collection of constraints on the relations between each two events, for example, whether two intervals overlap or not. One has to determine if there exists a realization of the events as intervals on the line, satisfying all the constraints. This is a generalization of the problem that Seymour Benzer faced trying to establish the linearity of genes, but at the time Shamir did not know about Benzer's work. He says:

> I presented my work on temporal reasoning at a workshop in 1990 and the late Gene Lawler told me this model fits perfectly the DNA mapping problem. I knew nothing about DNA at the time but started reading, and I also sent a note to Eric Lander describing the temporal reasoning result with the mapping consequences. A few weeks later Rutgers University organized a kickoff day on computational biology and brought Mike Waterman and Eric Lander who gave fantastic talks on the genome project and the computational challenges. Eric even mentioned our result as an important example of computer science contribution to the field. I think he was exaggerating quite a bit on this point, but the two talks sent me running to read biology textbooks and papers. I got a lot of help during the first years from my wife

> Michal, who happens to be a biologist and patiently answered all my trivial questions.

This chance meeting with Gene Lawler and Eric Lander converted Shamir into a bioinformatician and since 1991 he has been devoting more and more time to algorithmic problems in computational biology. He still keeps an interest in graph algorithms and often applies graph-theoretical techniques to biological problems. In recent years, he has tended to combine discrete mathematics techniques and probabilistic reasoning. Ron says:

> I still worry about the complexity of the problems and the algorithms, and would care about (and enjoy) the $\mathcal{NP}$-hardness proof of a new problem, but biology has trained me to be "tolerant" to heuristics when the theoretical computer science methodologies only take you so far, and the data require more.

Shamir was an early advocate of rigorous clustering algorithms in bioinformatics. Although clustering theory has been under development since the 1960s, his interest in clustering started in the mid-1990s as part of a collaboration with Hans Lehrach of the Max Planck Institute in Berlin. Lehrach is one of the fathers of the oligonucleotide fingerprinting technology that predates the DNA chips techniques. Lehrach had a "factory" in Berlin that would array thousands of unknown cDNA clones, and then hybridize them to $k$-mer probes (typically 8-mers or 9-mers). This was repeated with dozens of different probes, giving a fingerprint for each cDNA clone. To find which clones correspond to the same gene, one has to cluster them based on their fingerprints, and the noise level in the data makes accurate clustering quite a challenge.

It was this challenge that brought Shamir to think about clustering. As clustering theory is very heterogeneous and scattered over many different fields, he set out to develop his algorithm first, before learning the literature. Naturally, Shamir resorted to graph algorithms, modeled the clustered elements (clones in the fingerprinting application) as vertices, and connected clones with highly similar fingerprints by edges. It was a natural next step to repeatedly partition the graph based on minimum cuts (sets of edges whose removal makes the graph disconnected) and hence his first clustering algorithm was born. Actually the idea looked so natural to Shamir that he and his student Erez Hartuv searched the literature for a long time just to make sure it had not been published before. They did not find the same approach but found similar lines of thinking in works a decade earlier. According to Shamir,

> Given the ubiquity of clustering, I would not be surprised to unearth in the future an old paper with the same idea in archeology, zoology, or some other field.

Later, after considering the pros and cons of the approach, together with his student Roded Sharan, Shamir improved the algorithm by adding a rigorous probabilistic analysis and created the popular clustering algorithm CLICK. The publication of vast data sets of microarray expression profiles was a great opportunity and triggered Shamir to further develop PCC and CAST together with Amir Ben-Dor and Zohar Yakhini.

Shamir is one of the very few scientists who is regarded as a leader in both algorithms and bioinformatics and who was credited with an important breakthrough in algorithms even before he became a bioinformatician. He says:

> Probably the most exciting moment in my scientific life was my PhD result in 1983, showing that the average complexity of the Simplex algorithm is quadratic. The Simplex algorithm is one of the most commonly used algorithms, with thousands of applications in virtually all areas of science and engineering. Since the 1950s, the Simplex algorithm was known to be very efficient in practice and was (and is) used ubiquitously, but worst-case exponential-time examples were given for many variants of the Simplex, and none was proved to be efficient. This was a very embarrassing situation, and resolving it was very gratifying. Doing it together with my supervisors Ilan Adler (a leader in convex optimization, and a student of Dantzig, the father of the Simplex algorithm) and Richard Karp (one of the great leaders of theoretical computer science) was extremely exciting. Similar results were obtained independently at the same time by other research groups. There were some other exhilarating moments, particularly those between finding out an amazing new result and discovering the bug on the next day, but these are the rewards and frustrations of scientific research.

Shamir views himself as not committed to a particular problem. One of the advantages of scientific research in the university setting is the freedom to follow one's interests and nose, and not be obliged to work on the same problem for years. The exciting rapid developments in biology bring about a continuous flow of new problems, ideas, and data. Of course, one has to make sure breadth does not harm the research depth. In practice, in spite of this "uncommitted" ideology, Ron finds himself working on the same areas for several years, and following the literature in these fields a while later. He tends to work on several problem areas simultaneously but tries to choose topics that are overlapping to some extent, just to save on the effort of understanding and following several fields. Shamir says:

> In my opinion, an open mind, high antennae, a solid background in the relevant disciplines, and hard work are the most important elements of discovery. As a judge once said, "Inspiration only comes to the law library at 3 AM". This is even more true in science. Flexibility and sensitivity to the unexpected are crucial.

Shamir views the study of gene networks, that is, understanding how genes, proteins, and other molecules work together in concert, as a long-standing challenge.

## 10.13 Problems

**Problem 10.1**

Determine the number of different ways to partition a set of $n$ elements into $k$ clusters.

**Problem 10.2**

Construct an instance of the $k$-Means Clustering problem for which the Lloyd algorithm produces a particularly bad solution. Derive a performance guarantee of the Lloyd algorithm.

**Problem 10.3**

Find an optimal algorithm for solving the $k$-Means Clustering problem in the case of $k = 1$. Can you find an analytical solution in this case?

**Problem 10.4**

Estimate the number of iterations that the Lloyd algorithm will require when $k = 1$. Repeat for $k = 2$.

**Problem 10.5**

Construct an example for which the CAST algorithm does not converge.

**Problem 10.6**

How do you calculate the trimming parameter $\delta$ in ADDITIVEPHYLOGENY?

**Problem 10.7**

Prove that a connected graph in which the number of vertices exceeds the number of edges by 1 is a tree.

**Problem 10.8**

Some of 8 Hawaiian islands are connected by airlines. It is known one can reach every island from any other (probably with some intermediate stops). Prove that you can visit all islands making no more than 12 flights.

**Problem 10.9**

Show that any binary tree with $n$ leaves has $2n - 3$ edges.

**Problem 10.10**

How many different unrooted binary trees on $n$ vertices are there?

**Problem 10.11**

Prove that every binary tree has at least two neighboring leaves.

**Problem 10.12**

Design a backtracking procedure to reconstruct the optimal assignment of characters in the Sankoff algorithm for the Weighted Small Parsimony problem.

**Problem 10.13**

While we showed how to solve the Weighted Small Parsimony problem using dynamic programming, we did not show how to construct a Manhattan-like graph for this problem. Cast the Weighted Small Parsimony problem in terms of finding a path in an appropriate Manhattan-like directed acyclic graph.

**Problem 10.14**

The *nearest neighbor interchange distance* between two trees is defined as the minimum number of interchanges to transform one tree into another. Design an approximation algorithm for computing the nearest neighbor interchange distance.

**Problem 10.15**

Find two binary trees with six vertices that are the maximum possible nearest neighbor interchange distance apart from each other.