14 Genomics

We shall now explore how the techniques from the previous chapters can be used in studying the genome of a species. We assume here that we have available the so-called finalized genome assembly of a species. Ideally, this means the complete sequences of each chromosome in a species. However, in practice, this consists of chromosome sequences containing large unknown substrings, and some unmapped contigs/scaffolds.

To fix our mindset, we assume that the species under consideration is a diploid organism (like a human). We also assume that the assembled genome is concatenated into one long sequence T with some necessary markers added to separate its contents, and with some auxiliary bookkeeping data structures helpful for mapping back to the chromosome representation.

We start with a *peak detection* problem in Insight 14.1 that gives a direct connection to the segmentation problem motivating our study of hidden Markov models in Chapter 7. Then we proceed into variation calling, where we mostly formulate new problems assuming read alignment as input. The results of variant calling and read alignments are then given as the input to haplotype assembly.

Insight 14.1 Peak detection and HMMs

The *coverage profile* of the genome is an array storing for each position the amount of reads aligned to cover that position. In ChIP-sequencing and bisulfite sequencing only some parts of the genome should be covered by reads, so that clear peak areas should be noticeable in the coverage profile. In targeted resequencing, the areas of interest are known beforehand, so automatic detection of peak areas is not that relevant (although usually the targeting is not quite that accurate). Peak detection from a signal is a classical signal processing task, so there are many existing general-purpose peak detectors available. In order to use the machinery we developed earlier, let us consider how HMMs could be used for our peak detection task. The input is a sequence $S_{1..n}$ of numbers, where $s_j \geq 0$ gives the number of reads *starting* at position j in the reference. Our HMM has two states: `Background` and `Peak area`. Assume that r_b reads come from the background and r_p from peak areas, where $r = r_b + r_p$ is the number of reads. Assume that the background covers n_b positions and the peak areas cover n_p positions, where $n = n_b + n_p$ is the length of the genome. Then the probability that there are s_j reads starting at position j inside the peak area is given by the probability of an event in the binomial distribution

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

$$\mathbb{P}(s_j \mid j \in \texttt{Peak area}) = \binom{r_p}{s_j} \left(\frac{1}{n_p}\right)^{s_j} \left(\frac{n_p - 1}{n_p}\right)^{r_p - s_j}. \tag{14.1}$$

An analogous equation holds for positions j in the background area. To fix the HMM, one needs to set realistic a-priori values for transition probabilities so that switching from peak area to background, and vice versa, is expensive. One can use Baum–Welch or Viterbi training to find values for the parameters n_p, n_b, r_p, and r_b and for the transition probabilities. The peak areas found using ChIP-sequencing data should contain a common motif that explains the binding of the protein targeted by the experiment. *Motif discovery* algorithms are beyond the scope of this book; see references in the literature section.

14.1 Variation calling

We now recall some concepts from Chapter 1. A *reference* genome represents an average genome in a population, and it is of general interest to study what kind of variants are frequent in the population. However, one is also interested in the genome of a certain individual, called the *donor*. For example, in cancer genetics it is routine practice to sequence a family carrying an inherited disease, and to look for those variants unique to the family, when compared with the rest of the population. Such variants are also called *germline mutations*. *Somatic mutations* are those variations which do not affect germ cells. They sometimes affect the cell metabolism, and spread their effects to nearby cells, causing new somatic mutations that may, possibly together with existing germline mutations, cause cancer.

The process of identifying variants from sequencing data is called *variation calling*; the terms variant/variation detection/analysis are also frequently used. The approaches employed to discover these variants differ significantly depending on the scale of the sought-for variants. Most of them are based on the read alignment techniques we covered in Chapter 10, and on the subsequent analysis of the *read pileup*, that is, of the read alignments over a certain position in T. We now discuss some practical variation calling schemes.

14.1.1 Calling small variants

Let us first consider *single-nucleotide polymorphisms, SNPs*. Read pileup provides direct information about SNPs in the donor genome: if position j in T is covered by r_j reads, out of which p percent say that T contains a nucleotide a on position j, and the rest say that T contains a nucleotide b, one can reason whether this is because of heterozygous polymorphism or because of a measurement error. Measurement errors are typically easy to rule out; they are independent events, so the probability of observing many in the same position decreases exponentially: see Exercise 14.1.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

Hence, by sequencing with a high enough coverage, one can easily obtain the SNP profile of the donor genome with reasonable accuracy. This assumes that most reads are aligned to the correct position. The analysis can also take into account the mapping accuracy, to put more weight on trusted alignments.

The same straightforward idea can basically be applied to short indels, but more care needs to be taken in this case. First, if the read alignments are conducted in terms of the k-errors search problem, one can observe only indels of size at most k. Second, since the reads are aligned independently of each other, their alignments can disagree on the exact location of the indel. To tackle the latter problem, one can take the r_j reads covering position j and compute a multiple alignment; this phase is called *re-alignment* and it is likely to set the indels to a consensus location. This multiple alignment instance is of manageable size, so even optimal algorithms to each subproblem may be applied.

14.1.2 Calling large variants

Larger variations need a more global analysis of the read pileup. If there is a deletion in the donor genome longer than the error threshold in read alignment, then there should be an un-covered region in the read pileup with the same length as the deletion. If there is an insertion in the donor genome longer than the error threshold in read alignment, then there should be a pair of consecutive positions $(j, j+1)$ in the read pileup such that no read covers both j and $j+1$ as shown in Figure 14.1.

Moreover, there should be reads whose prefix matches some $T_{j'..j}$, and reads whose suffix matches some $T_{j+1..j''}$; in the case of deletions, both conditions should hold for some reads. To "fill in" the inserted regions, one can apply fragment assembly methods from Chapter 13: consider all the reads that are not mapped anywhere in the genome, together with substrings $T_{j-m+1..j}$ and $T_{j+1..j+m+1}$ of length m (m being the read length); try to create a contig that starts with $T_{j-m+1..j}$ and ends with $T_{j+1..j+m+1}$. This is just a simplistic view of indel detection, since in practice one must prepare for background noise.

Another approach for larger-scale variation calling is to use paired-end, or mate pair, reads. Consider paired-end read pairs $(L^1, R^1), (L^2, R^2), \ldots, (L^r, R^r)$ such that each read

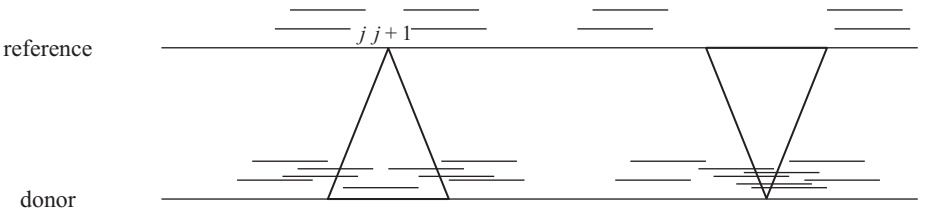


Figure 14.1 Analysis of read pileup for indels. On the left, an insertion in the donor; on the right, a deletion in the donor.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

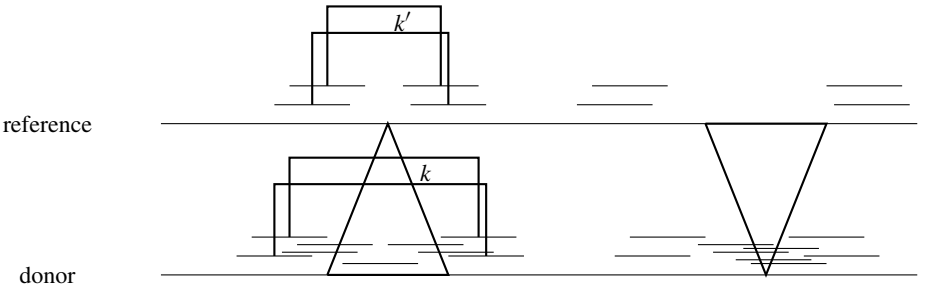


Figure 14.2 Analysis of the expected indel length. On the left, an insertion in the donor; on the right, a deletion in the donor.

L^i covers the same position j of the reference. Suppose that k is the expected distance between the input paired-end reads, with some variance, both known in advance from the library preparation protocol. One can compute the average, k', and the variance of the observed distances between the mapping positions of all L^i and R^i. The distance deviation can be tested to ascertain whether it is statistically significant; in such a case, the expected indel length is $|k - k'|$. This estimation is illustrated in Figure 14.2.

The length of the insertion is useful also for the fragment assembly approach mentioned above, since it imposes a constraint on the contig to be assembled. This problem is identical to the *gap-filling* problem, see Section 13.4.

In the case of insertions longer than the paired-end read length, one should be able to find the boundaries of the inserted area by analyzing read pairs having one end that is not mapped anywhere, but in this case the insert size remains unpredicted.

The third approach is to use split-read alignments: a read spanning an area deleted in the donor sequence could be aligned to the reference so that its prefix maps before the deleted area and its suffix maps after the deleted area. When many such split-read alignments vote for the same boundaries, one should be able to detect the deleted area accurately.

Let us now discuss how the paired-end approach gives rise to a graph optimization problem. Observe that in diploid organisms also indels can be homozygous or heterozygous, and, moreover, heterozygous indels can overlap. This means that two paired-end reads whose alignments overlap may in fact originate from different haplotypes. We wish to assign each paired-end read to support exactly one variant hypothesis, and to select a minimum set of hypotheses (the most parsimonious explanation) to form our prediction.

Assume we have a set H of hypotheses for the plausible variants, found by split-read mapping, by drops in read-pileup, or by paired-end alignments. Each hypothesis is a pair (j, g), where j is the starting position of the indel and g is its expected length. We say that an alignment of a paired-end read r *supports* a hypothesis $(j, g) \in H$, if the left part of r aligns left from j, the right part of r aligns right from j, and the difference between the alignment length and the expected indel size is g' ($g' = k - k'$ in Figure 14.2) such that $|g - g'| \leq t$, where t is a given threshold. We obtain a weighted bipartite graph

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

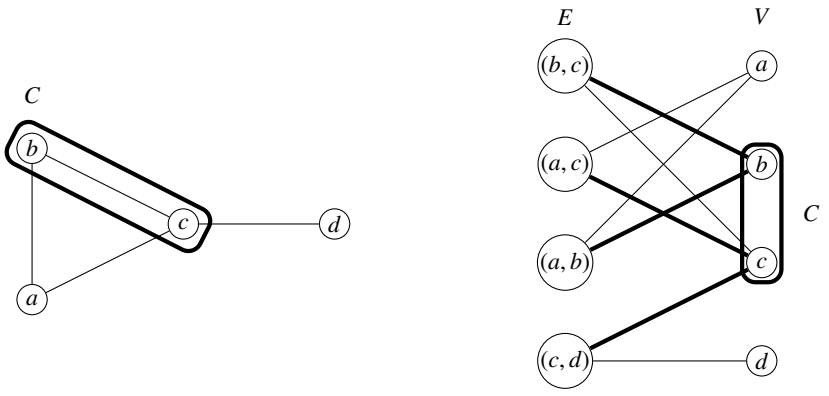


Figure 14.3 On the left, an undirected graph $G = (V, E)$ and a vertex cover C in G. On the right, the graph $G^* = (E \cup V, A)$ constructed in the proof of Theorem 14.1; a solution M for Problem 14.1 on G^* is highlighted.

$G = (R \cup H, E)$, where $r \in R$ is a paired-end read, $h = (j, g) \in H$ is a hypothesis, $(r, h) \in E$ if there is an alignment of r supporting hypothesis h, and the cost of the edge (r, h) is $c(r, h) = |g - g'|$, where g' is defined as above.

We start with a problem formulation (Problem 14.1) which captures the features of this problem. We show that Problem 14.1 is NP-hard, and then briefly discuss how this biological problem can be transformed into another NP-hard problem, called minimum-cost set cover. This alternative formulation is a standard problem, and can thus benefit from other results in the literature; see the further references in the last section of this chapter.

Problem 14.1 Minimum-cost minimum covering assignment

Given a bipartite graph $G = (R \cup H, E)$ and a cost function $c : E \to \mathbb{Q}_+$, find a subset $M \subseteq E$ of the edges of G such that

- each $r \in R$ is incident to exactly one $e \in M$, and
- the number of vertices of H incident to an edge of M is minimum.

Moreover, among all sets M satisfying the above two conditions, find one minimizing the total cost $\sum_{e \in M} c(e)$.

In practice, to make sure that our bipartite graph of reads and hypotheses always admits at least one solution for Problem 14.1, one can add a dummy hypothesis for each read, having a cost amounting to the sum of all existing edge costs; such costs allow the selection of these edges only when there is no solution for Problem 14.1 to the original input graph.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

THEOREM 14.1 *Problem 14.1 is NP-hard.*

Proof We reduce from the NP-hard vertex cover problem (recall Example 2.3 on page 16). Given an undirected graph $G = (V, E)$, create a bipartite incidence graph $G^* = (E \cup V, A)$ such that $A = \{(e, v) \mid e \in E \text{ is incident to } v \in V \text{ in } G\}$, and set the cost to all its edges to zero (see Figure 14.3). Given a set $M \subseteq A$, denote by $d_M(V)$ the number of vertices in V incident in G^* to an edge of M, namely

$$d_M(V) = |\{v \in V \mid \text{ there exists } (e, v) \in M\}|.$$

We claim that $G = (V, E)$ has a vertex cover of size at most k if and only if Problem 14.1 on G^* has a solution M satisfying $d_M(V) \leq k$.

For the forward implication, let $C \subseteq V$ be a vertex cover of G. Since C touches all edges of G in at least one endpoint, we can construct a solution M for Problem 14.1 on G^* by selecting, for every $e \in E$, one arbitrary edge $(e, v) \in A$ such that $v \in C$. Thus, $d_M(V) \leq |C|$.

For the reverse implication, let $M \subseteq A$ be a solution to Problem 14.1 on instance G^*, and let $C \subseteq V$ be the set of vertices in V incident to one element of M (observe that $|C| = d_M(V)$). Since each $e \in E$ is incident in G^* to one $a \in M$, we get that C is a vertex cover of G. □

Exercise 14.3 asks the reader to show that a greedy algorithm for this problem can produce a solution with a number of hypotheses greater by a factor $O(|H|)$ than the optimal one.

Observe that, if we relax Problem 14.1 by dropping the requirement that each read should support exactly only one hypothesis, then we obtain the following classical problem.

Problem 14.2 Minimum-cost set cover

Given a collection of subsets $H = \{H_1, \ldots, H_n\}$ of a set $\{r_1, \ldots, r_m\}$ and a cost function $c : \{H_1, \ldots, H_n\} \to \mathbb{Q}_+$, find a subset $C \subseteq H$ such that

- each r_i belongs to at least one H_j in C,

and it minimizes

$$\sum_{H_j \in C} c(H_j).$$

For our variant calling problem, we can assign as the cost for every hypothesis the sum of the costs of the reads supporting it. This problem admits a polynomial-time algorithm producing a solution whose total cost is at most $O(\log|H|)$ times the optimal one; see Exercise 14.4 and the literature section for references to the minimum-cost set cover problem.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

14.2 Variation calling over pan-genomes

In Section 10.7 we discussed two approaches for aligning reads, by taking into account variants already discovered in a population. Let us now consider two approaches to how a donor sequence can be predicted, on the basis of these alignments. For both of them, the idea is to first predict which known variants from the population are included in the donor sequence, and then predict the donor-specific variants. This donor-specific prediction is done using the scheme from the previous section, by taking as reference the predicted donor sequence.

14.2.1 Alignments on a set of individual genomes

Assume that we have the read alignments on top of a collection of d reference sequences, whose multiple alignment we already know. Namely, we have a $d \times n$ matrix as defined in Section 6.6. We can then create another $d \times n$ matrix C, which contains now the read coverage counts; that is, each $c_{i,j}$ tells us how many reads cover the ith genome at position j of the multiple alignment. Figure 14.4 gives an example of the coverage counts.

Consider the simple strategy of predicting a donor sequence by choosing, at each column j, the row i with the maximum value $c_{i,j}$. Extracting the corresponding nucleotides from multiple alignment and removing the gap symbols gives us a predicted donor genome.

Observe that such a predicted genome recombines the underlying sequences arbitrarily, by choosing the recombination having most support by read alignments. However, in order to model more closely the biological process, we can limit the number of recombinations by a given parameter. This leads to Problem 14.3 below, whose setting is illustrated in Figure 14.4.

Problem 14.3 Heaviest path with recombination constraint

Given a $d \times n$ matrix C, with $c_{i,j} \geq 0$ for all i,j, and a threshold $K < n$, find a path in C that traverses at most K rows of C, namely a sequence P of cells of C, $P = (i_1, 1), (i_1, 2), \ldots, (i_1, j_1), (i_2, j_1 + 1), (i_2, j_1 + 2), \ldots, (i_2, j_2), \ldots, (i_K, j_{K-1} + 1), (i_K, j_{K-1} + 2), \ldots, (i_K, n)$, maximizing

$$\sum_{(i,j) \in P} c_{i,j}.$$

Problem 14.3 can be solved by dynamic programming, as follows. Let $v(i, j, k)$ denote the value of the optimum solution for the submatrix $C_{1..i,1..j}$, among all paths recombining rows $k - 1$ times. The matrix v can be computed with the following recurrence:

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

1244444443333225778888888887744232112
6677788998887664232222222333333211111
3334444444333110000011113333225588899

Figure 14.4 Illustration of a heaviest path in a matrix C with recombination constraint $K = 3$. Each row of the matrix stands for a different reference sequence, and each column corresponds to a position in the multiple alignment of the three reference sequences. The pieces corresponding to the optimal path through the matrix are underlined.

$$v(i,j,k) = \max\left(v(i,j-1,k) + c_{i,j}, \max\{v(i',j-1,k-1) + c_{i,j} \mid 1 \leq i' \leq d \text{ and } i' \neq i\}\right), \tag{14.2}$$

where we initialize $v(i,1,1) = c_{i,1}$, for all $1 \leq i \leq d$. The value of the optimal solution for the entire matrix C is obtained as

$$\max\{v(i,n,K) \mid 1 \leq i \leq d\}.$$

Under a suitable evaluation order, the matrix v can be computed in time $O(dnK)$. The corresponding optimal path can be extracted by a traceback in the matrix v. A straightforward implementation of this traceback can be implemented in time $O(dnK)$ and space $O(dnK)$. However, using the same idea as the reader was asked for in Exercise 7.2 (about a space-efficient implementation of the Viterbi algorithm), this traceback can be implemented with the same time complexity, but using only $O(d\sqrt{n}K)$ space. We have thus proved the following.

THEOREM 14.2 *Problem 14.3 can be solved in $O(dnK)$ time and $O(d\sqrt{n}K)$ space.*

14.2.2 Alignments on the labeled DAG of a population

Recall the approach to pan-genome indexing in which the reference, and the known variants, are represented as a labeled DAG G (illustrated in Figure 10.6 on page 215). When extracting the path to which each read is aligned, as shown in Section 10.7, we can increase a counter $c(v)$, for each vertex v covered by the read.

Observe that in the previous section, since we had available each individual genome of the population, we could impose a limit on the number of recombinations. In the case of G, this information is no longer available, so we also assume that the donor sequence is a heaviest path in G (from a source to a sink), namely one maximizing the sum of the weights of its vertices.

The dynamic programming recurrences that we derived for the shortest-path problem on a DAG with arc weights (Problem 4.1) carry over to this problem, resulting in an algorithm running in time linear in the number of arcs of G. Equivalently, one can apply any algorithm for solving Problem 4.1, by creating a DAG G^* obtained from G by subdividing each vertex v into two vertices v_{in} and v_{out}, with all in-neighbors of v now in-neighbors of v_{in}, and all out-neighbors of v now out-neighbors of v_{out}, and adding the arc (v_{in}, v_{out}) with weight $-c(v)$. All other arcs of G^* have weight equal to zero. A shortest path in the DAG G^* from a source to a sink is a heaviest path in G.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

14.2.3 Evaluation of variation calling results

Variation calls can be validated by using targeted sequencing, but this is an expensive process, and partly manual wet-lab work. Hence, simulation studies mimicking a real setting are needed for guiding the variation calling tool development.

For such a study, one can create a diploid ground-truth by taking a reference sequence, create homozygous random mutations to it, and then obtain two haplotypes by carrying out this process twice independently, in order to create the heterozygous mutations. The result can be represented as a pair-wise alignment of an artificial diploid individual. One can then generate reads from these two sequences from random positions, in order to mimic sequencing.

In a pan-genomic setting, one can add another layer at the beginning by taking a random sample of already observed variants in the population. In this way the generated ground-truth models the likely germline mutations and some rarer germline mutations, as well as somatic mutations. We discuss in Insight 14.2 how one could evaluate the predictions of several variation calling tools with such data.

Insight 14.2 Variation calling evaluation and DAG alignment

Consider a tool producing a set of variant predictions and the evaluation problem of checking how close the predictions are to the ground-truth artificially generated diploid genome. A one-to-one mapping of these predictions to the true variants is not always possible, since predictions may be inaccurate (especially for longer indels). One way to overcome this difficulty is to apply the predicted variants to the reference to form a predicted haploid donor genome. This predicted donor sequence does not necessarily need to be close to either of the two sequences forming the diploid ground-truth, but it should be close to some recombination of them; otherwise it is a bad prediction.

That is, we should align a haploid prediction to a diploid ground-truth allowing recombinations. We solved this problem already in Section 6.6.5: we can convert the diploid pair-wise alignment into a DAG, and represent the haploid as a trivial DAG (one path from source to sink labeled by the haploid). Although the running time of this approach is quadratic, in practice, with some assumptions on the underlying DAG, one can apply the shortest-detour method of Section 6.1.2 to obtain an $O(dn)$ time algorithm, where d is the edit distance from the predicted haploid to the best recombination of the diploid ground-truth: see Exercise 14.6.

14.3 Haplotype assembly and phasing

With diploid organisms, one is interested not only in discovering variants, but also in discovering to which of the two haplotypes each variant belongs. One would thus like to identify the variants which are co-located on the same haplotype, a process called *haplotype phasing*. The variants present in an individual which are clearly homozygous

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

can be trivially assigned to both haplotypes. Thus, in this section we limit ourselves to heterozygous variants, and, furthermore, just to SNP variants.

Thanks to high-throughput sequencing technologies, we can guide this process by read alignments and read overlaps. Consider the problem formulation below, and see also Example 14.5.

Problem 14.4 Haplotype assembly under minimum error correction

Given an $r \times s$ matrix M, with values $\{0, 1, -\}$, where reads are rows, columns are SNPs, and

$$m_{i,j} = \begin{cases} 0, & \text{if the } i\text{th read does not support the } j\text{th SNP,} \\ 1, & \text{if the } i\text{th read supports the } j\text{th SNP,} \\ -, & \text{if the } i\text{th read does not overlap the } j\text{th SNP,} \end{cases}$$

find

- a partition of the rows of M (the reads) into two subsets R^0 and R^1, and
- two binary sequences A and B of length s (the two haplotypes),

which minimize the number of bit flips in M that are required to make the reads in R^0 compatible with A, and the reads in R^1 compatible with B.

We say that a read i is compatible with a haplotype $X = x_1 \cdots x_s$ if, for all $j \in \{1, \ldots, s\}$, $m_{i,j} = -$ or $m_{i,j} = x_j$.

Observe that, for clarity, in Problem 14.4 we ask both for a partition of the reads and for the two haplotypes. However, having an optimal partition (R^0, R^1) of the rows of M, we can easily find the two optimal haplotypes in a column-wise manner. For every $j \in \{1, \ldots, s\}$, we either set $A[j] = 0, B[j] = 1$, by flipping all bits in R^0 on column j equal to 1 and all bits in R^1 on column j equal to 0, thus with a total of

$$|\{i \in R^0 \mid m_{i,j} = 1\}| + |\{i \in R^1 \mid m_{i,j} = 0\}| \tag{14.3}$$

bit flips, or we set $A[j] = 1, B[j] = 0$ with a total of

$$|\{i \in R^0 \mid m_{i,j} = 0\}| + |\{i \in R^1 \mid m_{i,j} = 1\}| \tag{14.4}$$

bit flips. We set $A[j]$ and $B[j]$ by checking which of the two costs (14.3) or (14.4) is minimum. Moreover, it will be convenient to denote the minimum of the two by $c(R^0, R^1, j)$, namely

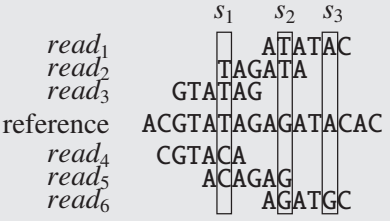
$$c(R^0, R^1, j) = \min \Big(|\{i \in R^0 \mid m_{i,j} = 1\}| + |\{i \in R^1 \mid m_{i,j} = 0\}|, \\ |\{i \in R^0 \mid m_{i,j} = 0\}| + |\{i \in R^1 \mid m_{i,j} = 1\}| \Big).$$

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

Conversely, having the two haplotypes A and B, it is easy to find the optimal partition (R^0, R^1) row-wise by setting, for each $i \in \{1, \ldots, r\}$,

$$i \in \begin{cases} R^0, & \text{if } |\{j \mid m_{i,j} \neq - \text{ and } m_{i,j} \neq A[j]\}| \leq |\{j \mid m_{i,j} \neq - \text{ and } m_{i,j} \neq B[j]\}|, \\ R^1, & \text{otherwise.} \end{cases}$$

Example 14.5 Consider the following setting in which six reads have been aligned to a reference, resulting in the SNPs `C` at position s_1, `T` at position s_2, and `G` at position s_3. We assume the two unknown haplotypes to be `ACGTATAGATATACAC` (from which the three reads on the top originate) and `ACGTACAGAGATGCAC` (from which the three reads on the bottom originate).



We construct the following matrix M:

$$M = \begin{pmatrix} - & 1 & 0 \\ 0 & 1 & - \\ 0 & - & - \\ 1 & - & - \\ 1 & 0 & - \\ - & 0 & 1 \end{pmatrix},$$

where $m_{i,j} = 0$ tells that the ith read does not support the jth SNP, $m_{i,j} = 1$ tells that the ith read supports the jth SNP, and $m_{i,j} = -$ tells that the ith read does not overlap the jth SNP.

The two haplotypes are 010 and 101, and they correspond to assigning the first three reads (rows) to the first haplotype, and the other three reads (rows) to the second haplotype; that is, $R^0 = \{1, 2, 3\}$ and $R^1 = \{4, 5, 6\}$. In this case, no bit flips in M are needed, since all the reads (rows) are compatible with one of the two haplotypes.

Haplotype phasing was known to be a hard problem even before the development of high-throughput sequencing. Unfortunately, even the above haplotype assembly formulation based on read alignments remains so.

THEOREM 14.3 *Problem 14.4 is NP-hard.*

Proof We reduce from the NP-hard problem MAX-CUT. In this problem, one is given an undirected graph $G = (V, E)$ and is asked to find a subset C of vertices that maximizes the number of edges between C and $V \setminus C$, that is, edges having one endpoint in C and the other one outside C.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

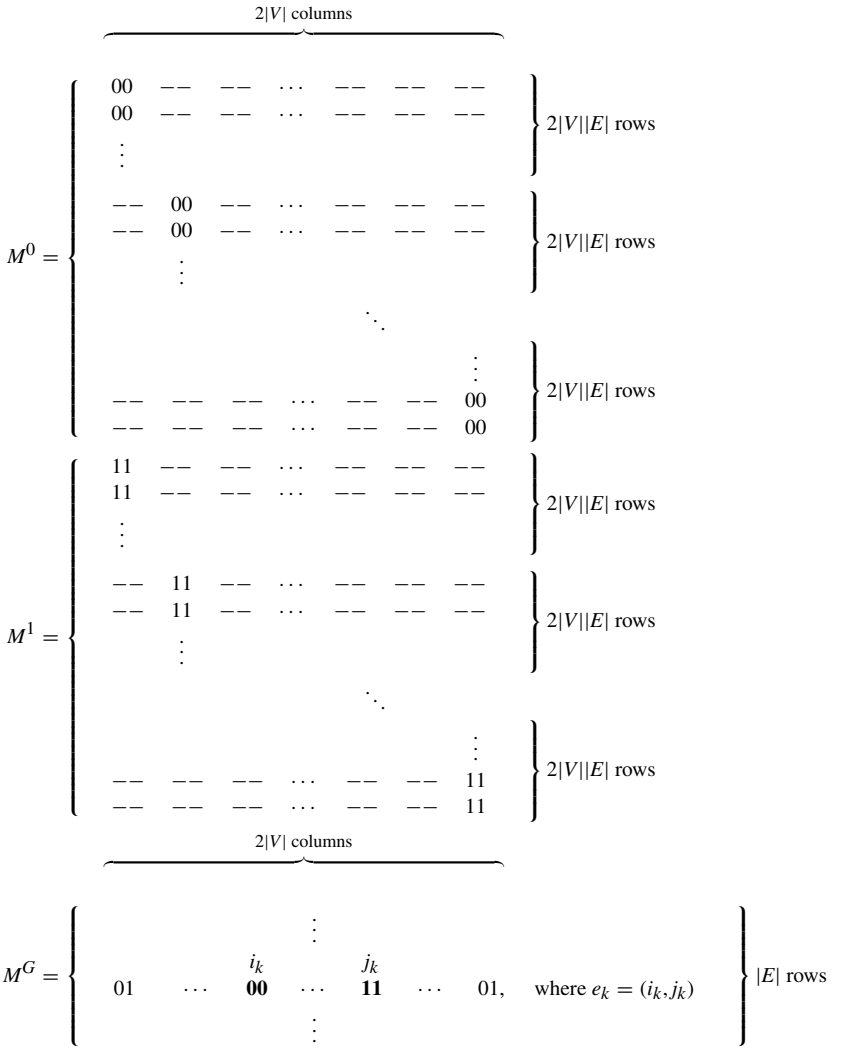


Figure 14.5 The three binary matrices needed in reducing the MAX-CUT problem to Problem 14.4.

Given a graph G as input for the MAX-CUT problem, we construct a matrix M made up of the rows of the three matrices M^0, M^1, and M^G shown in Figure 14.5, (that is, M has $2|V|$ columns and $2|V|^2|E| + 2|V|^2|E| + |E|$ rows).

The matrix M^G has a row for each edge $(i,j) \in E$. To simplify the notation, let $V = \{1, \ldots, |V|\}$ and $E = \{e_1, \ldots, e_{|E|}\}$. For each $e_k \in E$ with $e_k = (i,j)$, $i < j$, we set

- $m^G_{k,2i-1} = m^G_{k,2i} = 0$,
- $m^G_{k,2j-1} = m^G_{k,2j} = 1$,
- $m^G_{k,2t-1} = 0$ and $m^G_{k,2t} = 1$, for all $t \in \{1, \ldots, |V|\} \setminus \{i,j\}$.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

We will show that $G = (V, E)$ admits a subset $C \subseteq V$ with t edges between C and $V \setminus C$ if and only if the number of flips in an optimal solution for M is $|V||E| - 2t$. Thus, finding a maximum cut in G is equivalent to finding a solution with the minimum number of flips for M.

For the forward implication, let $C \subseteq V$ be such that there are t edges between C and $V \setminus C$. Define the haplotypes A and B as

$$A[2i-1] = \begin{cases} A[2i] = 0, B[2i-1] = B[2i] = 1, & \text{if vertex } i \in C, \\ A[2i] = 1, B[2i-1] = B[2i] = 0, & \text{otherwise.} \end{cases}$$

Observe first that each row of M^0 and M^1 is compatible with either A or B, without flipping any bits. Moreover, for every edge $e_k = (i, j)$, with $i < j$, of G the following statements apply.

- If e_k is between C and $V \setminus C$, then the kth row of M^G needs exactly $|V| - 2$ bit flips to be compatible with one of A or B, one bit for exactly one of the positions $m_{k,2t-1}$ or $m_{k,2t}$, for all $t \in \{1, \ldots, |V|\} \setminus \{i, j\}$.
- If e_k is not between C and $V \setminus C$, then assume, without loss of generality, that $i, j \in C$ (the case $i, j \in V \setminus C$ is symmetric). Since by construction $m_{k,2i-1} = m_{k,2i} = 0$, and $m_{k,2j-1} = m_{k,2j} = 1$, but $A[2i-1] = A[2i] = 0 = A[2j-1] = A[2j]$, we need to flip $m_{k,2j-1}$ and $m_{k,2j}$. Therefore, the kth row of M^G needs $|V|$ bit flips, one bit for exactly one of the positions $m_{k,2t-1}$ or $m_{k,2t}$, for all $t \in \{1, \ldots, |V|\} \setminus \{i, j\}$, and two more bit flips for $m_{k,2j-1}$ and $m_{k,2j}$.

Thus, we have constructed a solution for M requiring in total $|V||E| - 2t$ bit flips.

The reverse implication is left as an exercise for the reader (Exercise 14.7) □

We will now present a practical algorithm for solving Problem 14.4. Onwards, we are concerned with finding an optimal partition (R^0, R^1) of the rows of M, since, as explained on page 316, we can easily derive the two optimal haplotypes from an optimal partition.

We start with the simple strategy of trying out all possible ways of partitioning the rows of M, and choosing the partition requiring the minimum number of bit flips. Given such a partition (R^0, R^1), let $D(R^0, R^1, j)$ denote the minimum number of bit flips needed for making the first j columns of M compatible with two haplotypes. Using the notation $c(R^0, R^1, j)$ introduced on page 316, we can clearly express D column-wise as

$$D(R^0, R^1, j) = D(R^0, R^1, j-1) + c(R^0, R^1, j). \tag{14.5}$$

Such an algorithm has complexity $O(2^r rs)$, since we try out all of the 2^r partitions (R^0, R^1) of the rows of M, and then evaluate them by computing $D(R^0, R^1, s)$ in time $O(rs)$ (computing $c(R^0, R^1, j)$ takes time $O(r)$).

Let us see now how such an algorithm can be refined for the practical features of the haplotype assembly problem. Observe that the number of reads overlapping a given SNP j (that is, having a 0 or a 1 on column j) is much smaller than the total number r of reads. We will obtain an algorithm of complexity $O(2^K(2^K + r)s)$, where K is the maximum coverage of an SNP. Such an algorithm is called *fixed-parameter tractable*,

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

with the maximum coverage as a parameter, since, if we assume K to be bounded by a constant, then we have a polynomial-time algorithm.

The cost of making a column j of M compatible with two haplotypes does not depend on the rows of M having a $-$. Call such rows *inactive* at column j, and call the rows having a 0 or a 1 *active* at column j. Therefore, for each column j, we can enumerate over all partitions only of its active rows. The cost of making the jth column, whose active rows are partitioned as (S^0, S^1), compatible with two haplotypes is obtained just as before, and we analogously denote it as $c(S^0, S^1, j)$.

We also analogously denote by $D(S^0, S^1, j)$ the minimum number of bit flips needed to make the first j columns of M compatible with two haplotypes, with the constraint that the active rows at the jth column are partitioned as (S^0, S^1). In order to express $D(S^0, S^1, j)$ depending on $D(S^0, S^1, j-1)$, observe that some rows are active both in column j and in column $j-1$. Denote by $U(j)$ the rows of M active both in column $j-1$ and in column j. Thus, the cost $D(S^0, S^1, j)$ depends only on those partitions (Q^0, Q^1) of the $(j-1)$th column which partition the rows in $U(j)$ in the same manner as (S^0, S^1). We can thus write

$$D(S^0, S^1, j) = \min\{D(Q^0, Q^1, j-1) \mid U(j) \cap S^0 \subseteq Q^0 \text{ and } U(j) \cap S^1 \subseteq Q^1\} + c(S^0, S^1, j), \tag{14.6}$$

where we take $D(S^0, S^1, 1) = c(S^0, S^1, 1)$. The number of bit flips in a solution to Problem 14.4 is obtained as (see Example 14.6)

$$\min\{D(S^0, S^1, s) \mid (S^0, S^1) \text{ is a partition of the rows active at the last column } s\}. \tag{14.7}$$

An optimal solution can be traced back by starting with the partition minimizing (14.7), and then iteratively adding to the two sets of this partition the sets in the partition minimizing (14.6). Since computing $c(S^0, S^1, j)$ takes $O(r)$ time, the number of partitions of the rows active at any column is at most 2^K, and, for every such partition in a column j, we consider all $O(2^K)$ partitions of the rows active in column $j-1$, we have the following result.

THEOREM 14.4 *Problem 14.4 can be solved in $O(2^K(2^K + r)s)$ time, where K is the maximum coverage of an SNP.*

Exercise 14.8 asks the reader to show that this algorithm can be implemented to run in time $O(2^K s)$. Observe that a direct implementation of this algorithm requires memory $O(2^K s)$, so that every partition of the active rows is stored for every column; see Exercise 14.9 for an implementation of this algorithm using memory $O(2^K\sqrt{s})$. Exercise 14.10 asks the reader to modify this algorithm for solving Problem 14.4 in time $O(K^T s)$, where T denotes the maximum number of bit flips required in any column in an optimal solution. Observe that in practice T is usually rather small.

Moreover, Problem 14.4 can be generalized so that every cell of M also has a cost of flipping (in practice, the confidence that the prediction is correct). Exercise 14.11 asks the reader which changes are needed to the above algorithm for solving this weighted problem.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

Example 14.6 Consider the matrix

$$M = \begin{pmatrix} - & 1 & 0 \\ 0 & 1 & - \\ 0 & - & - \\ 1 & - & - \\ 1 & 1 & 1 \\ - & - & 1 \end{pmatrix},$$

which is obtained from the one in Example 14.5 by changing the entries $m_{5,2}$, $m_{5,3}$, and $m_{6,2}$. Note that $U(2) = \{2, 5\}$ and $U(3) = \{1, 5\}$. For the first column, we have

$$\begin{aligned}
D(\emptyset, \{2,3,4,5\}, \mathbf{1}) &= 2 = D(\{2,3,4,5\}, \emptyset, \mathbf{1}),\\
D(\{2\}, \{3,4,5\}, \mathbf{1}) &= 1 = D(\{3,4,5\}, \{2\}, \mathbf{1}),\\
D(\{3\}, \{2,4,5\}, \mathbf{1}) &= 1 = D(\{2,4,5\}, \{3\}, \mathbf{1}),\\
D(\{4\}, \{2,3,5\}, \mathbf{1}) &= 1 = D(\{2,3,5\}, \{4\}, \mathbf{1}),\\
D(\{5\}, \{2,3,4\}, \mathbf{1}) &= 1 = D(\{2,3,4\}, \{5\}, \mathbf{1}),\\
D(\{2,3\}, \{4,5\}, \mathbf{1}) &= 0 = D(\{4,5\}, \{2,3\}, \mathbf{1}),\\
D(\{2,4\}, \{3,5\}, \mathbf{1}) &= 2 = D(\{3,5\}, \{2,4\}, \mathbf{1}),\\
D(\{2,5\}, \{3,4\}, \mathbf{1}) &= 2 = D(\{3,4\}, \{2,5\}, \mathbf{1}).
\end{aligned}$$

For the second column, we have

$$\begin{aligned}
D(\emptyset, \{1,2,5\}, \mathbf{2}) &= \min(D(\emptyset, \{2,3,4,5\}, \mathbf{1}),\\
&\qquad D(\{3\}, \{2,4,5\}, \mathbf{1}),\\
&\qquad D(\{4\}, \{2,3,5\}, \mathbf{1}),\\
&\qquad D(\{3,4\}, \{2,5\}, \mathbf{1})) + 0 = 1 = D(\{1,2,5\}, \emptyset, \mathbf{2}),\\
D(\{5\}, \{1,2\}, \mathbf{2}) &= \min(D(\{3,4,5\}, \{2\}, \mathbf{1}),\\
&\qquad D(\{5\}, \{2,3,4\}, \mathbf{1}),\\
&\qquad D(\{4,5\}, \{2,3\}, \mathbf{1}),\\
&\qquad D(\{3,5\}, \{2,4\}, \mathbf{1})) + 1 = 1 = D(\{1,2\}, \{5\}, \mathbf{2}),\\
D(\{1\}, \{2,5\}, \mathbf{2}) &= 1 + 1 = D(\{2,5\}, \{1\}, \mathbf{2}),\\
D(\{2\}, \{1,5\}, \mathbf{2}) &= 0 + 1 = D(\{1,5\}, \{2\}, \mathbf{2}).
\end{aligned}$$

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

For the third column, we have

$$\begin{aligned} D(\emptyset, \{1,5,6\}, \mathbf{3}) &= \min(D(\emptyset, \{1,2,5\}, \mathbf{2}), \\ &\qquad D(\{2\}, \{1,5\}, \mathbf{2})) + 1 = 2 = D(\{1,5,6\}, \emptyset, \mathbf{2}), \\ D(\{1\}, \{5,6\}, \mathbf{3}) &= \min(D(\{1,2\}, \{5\}, \mathbf{2}), \\ &\qquad D(\{1\}, \{2,5\}, \mathbf{2})) + 0 = 1 = D(\{5,6\}, \{1\}, \mathbf{2}), \\ D(\{5\}, \{1,6\}, \mathbf{3}) &= 1 + 1 = D(\{1,6\}, \{5\}, \mathbf{3}), \\ D(\{6\}, \{1,5\}, \mathbf{3}) &= 1 + 1 = D(\{1,5\}, \{6\}, \mathbf{3}). \end{aligned}$$

The optimal partition $(\{1,2,3\}, \{4,5,6\})$ has cost $D(\{1\}, \{5,6\}, \mathbf{3}) = 1$, from flipping entry $m_{5,2}$.

14.4 Literature

We omitted algorithms for motif discovery related to the peak area analysis of Insight 14.1. The foundations of such algorithms are covered in a dedicated book by Parida (2007). A similar approach to ChIP-seq peak detection (using more complex distributions) is proposed in Qin *et al.* (2010): see the references therein for alternative approaches.

The basic approach of read filtering, read alignment, local re-alignment, variant calling from read pileup, and variant annotation, is more or less the standard workflow applied in biomedical laboratories. As is typical for such workflows, the choice of the software for implementing each subroutine is an art of engineering, with best-practice recommendations for each study case in question. From an algorithmic perspective, the key contributions are inside each subtask, and the literature for those can be found from previous chapters. For an overview of the best practices, there are numerous surveys and experimental tool comparisons available: see for example Pabinger *et al.* (2013).

Calling larger variants has been more on the focus of the algorithmically oriented bioinformatics community. Our formulation differs slightly from the literature. As far as we know, the covering assignment problem has not been formulated before, but instead some other routes to the set cover relaxation exist (Hormozdiari *et al.* 2009). See also the monograph by Vazirani (2001) for further references to the set cover problem. The hypotheses we formulated are usually defined as maximal cliques in an interval graph or some other graph defined by read alignment overlaps; see Hormozdiari *et al.* (2009), Marschall *et al.* (2012), and the references therein.

The pan-genome variant calling covered here breaks the standardized workflow, giving also algorithmically more interesting questions to study. As far as we know, variant calling from alignments to multiple genomes has not been seriously considered in the literature, except for some experiments in articles focusing on the alignment problem on multiple genomes. Our approach follows that of ongoing work (Valenzuela *et al.* 2015).

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

Haplotyping is a rich area of algorithm research. Here, we focused on the haplotype assembly formulation, since it is the closest to high-throughput sequencing. The NP-hardness proof is from Cilibrasi *et al.* (2005), and the dynamic programming algorithm is from Patterson *et al.* (2014); notice that they obtain a better complexity, see Exercise 14.8.

Exercises

14.1 Calculate the probability of an SNP given a read pileup taking into account the measurement errors.

14.2 *Variant annotation* is the process of assigning a function for each detected variant. For example, a 1-base deletion inside an exon creates a frame-shift and may cause an abnormal protein product to be translated. Consider different variants that might appear and think about why some variants can be called *silent mutations*. Browse the literature to find out what *nonsense mutations*, *missense mutations*, and *regulatory variants* are.

14.3 A greedy algorithm to solve Problem 14.1 is to start with empty E', choose $h \in H$ with most incoming edges not in E', add those unmatched edges to E', and iterate the process until all $r \in R$ are incident to an edge in E'. Show that this can produce a solution whose size is $O(|H|)$ times the optimal solution size.

14.4 Consider the following algorithm for the minimum-cost set cover problem. Start with an empty C and iteratively add to C the most cost-effective set H_i, that is, the set H_i maximizing the ratio between $c(H_i)$ and the number of un-covered elements of $\{r_1, \ldots, r_m\}$ that H_i covers. This is repeated until all elements are covered by some set in C. Show that this algorithm always produces a solution whose cost is $O(\log m)$ times the cost of an optimal solution.

14.5 Show that Problem 14.3 can be reduced to the particular shortest-path problem on DAGs from Exercise 4.16.

14.6 Under what assumptions does the shortest-detour algorithm of Section 6.1.2 applied to aligning haploid prediction to labeled DAG of diploid ground-truth give $O(dn)$ running time, where d is the resulting edit distance and n is the maximum of the DAG size and haploid length?

14.7 Complete the proof of Theorem 14.3, by proving that, if A and B are two haplotypes obtained with $|V||E| - 2t$ bit flips from M, then the set $C = \{i \mid A[2i-1] = A[2i] = 0\} \subseteq V$ has the property that there are t edges between C and $V \setminus C$. Obtain this proof by showing the following intermediary properties:

- $A[2i-1] = A[2i]$, for all $i \in \{1, \ldots, |V|\}$;
- $B[2i-1] = B[2i]$, for all $i \in \{1, \ldots, |V|\}$;
- A is the bit-wise complement of B.

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press

14.8 Show that the algorithm given for the haplotype assembly in the framework of the minimum error correction problem can be optimized to run in $O((2^K + r)s)$ time. How could you get the time complexity down to $O(2^K s)$?

14.9 Show how the algorithm given for the haplotype assembly in the minimum error correction problem can be implemented to use $O(2^K \sqrt{s})$ memory, while maintaining the same asymptotic time complexity. *Hint.* Use the same idea as in Exercise 7.2.

14.10 Let T denote the maximum number of bit flips needed in any column of the matrix M in an optimal solution to Problem 14.4. Show that the algorithm given for this problem can be modified to run in time $O((K^T + r)s)$, and even in time $O(K^T s)$ (see also Exercise 14.8 above).

14.11 Consider the problem of haplotype assembly under minimum error correction in which each cell of M has also a cost of flipping, and one is looking for a partition of the rows of M that is compatible with two haplotypes and minimizes the sum of the costs of all bit flips. What changes to the algorithm we presented are needed in order to solve this more general problem with the same time complexity?

https://doi.org/10.1017/CBO9781139940023.016 Published online by Cambridge University Press