Chapter 4
Fast Alignment

4.1 Alignment with k Errors

Exact matching is fast and uses little memory, while filling in alignment matrices is slow and memory consuming. Of these two central resources, time and memory, memory is often the more limiting: it is usually possible to wait a bit longer for a computation to finish, while a computation that does not fit into memory cannot even start. However, waiting for a calculation to complete is also undesirable, not least because computations that finish "instantaneously" can become part of larger, new applications. Useful programs are efficient.

The central insight for speeding up alignment is that the vast majority of cells in an alignment matrix are nowhere near an optimal path. But how can the "promising" parts of an alignment matrix be identified? The answer is, cheap exact matching. The k-error alignment method makes direct use of this idea [8]: Say we are looking for a short query sequence, Q, in a long subject sequence, S, and Q contains a single error. This could be an insertion, a deletion, or a mismatch; in the example shown in Fig. 4.1a it is a deletion, marked by a Δ.

If Q is divided into two fragments, a and b, the error is either located in segment a, as in Fig. 4.1a, or b, but not in both. Next, the algorithm searches for a and b in S and finds b (Fig. 4.1b). This match anchors the alignment matrix, which occupies $(|Q| + 1)^2$ cells rather than $(|Q| \times |S|)$ (Fig. 4.1c).

Put more generally, k-error alignment begins by choosing an upper limit for the number of errors the final alignment may contain, k. Then Q is divided into $k + 1$ fragments of equal length, each of which is searched exactly in S. Whenever a match is found, an alignment matrix is constructed with dimension $O((|Q| + k)^2)$, which is filled in by dynamic programming to check whether it contains a k-error match between Q and S.

As k grows, the fragment size decreases. Short fragments can match random positions in the subject making the checking phase potentially *more* time consuming than filling in the full alignment matrix. However, for small k the method works well in practice as we shall see in this section.

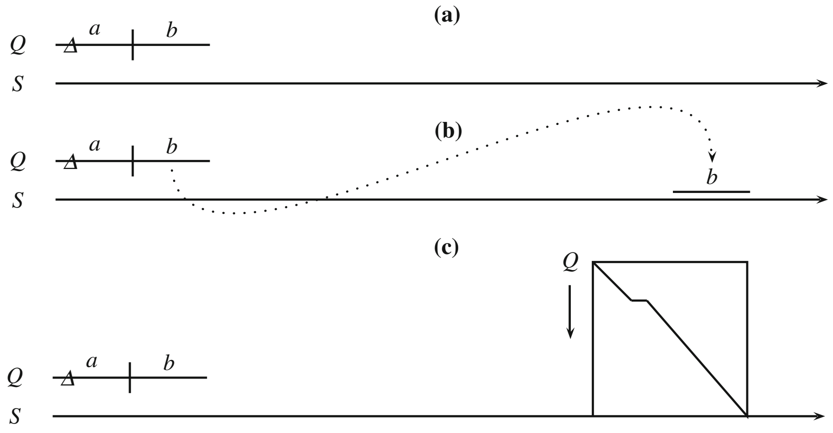


Fig. 4.1 The k-error alignment method comprises three steps: Division of the query into $k + 1$ contiguous fragments (**a**); exact search for the fragments (**b**), and checking whether or not a k-error alignment has been found by filling in the dynamic programming matrix anchored by the exact match (**c**)

New Concept

Name	Comment
k-error alignment	speed up alignment by exact matching

New Programs

Name	Source	Help
`kerror`	book website	`kerror -h`
`mutator`	book website	`mutator -h`

Problem 235 Create a new directory for this section, `KerrorAlignment`, and change into it. Copy the sequence of `dmAdhAdhdup.fasta`, cut out positions 2301-2400, and save the resulting sequence fragment in `dmAdhFrag.fasta`. Check the correct region was cut out by matching the fragment back onto `dmAdhAdhdup.fasta` using `keywordMatcher`.

Problem 236 Mutate position 10 in `dmAdhFrag.fasta` using the program `mutator` and save the mutated sequence in `dmAdhFrag2.fasta`. Check the mutation appeared at the expected position using `gal`. Then compare the mutated fragment to the original full sequence to confirm the mutated fragment does not match exactly any more (`keywordMatcher`), but the inexact match remained intact (`lal`).

Problem 237 The program `kerror` implements k-error matching. Use it to locate the error-free fragment in the original full sequence. Then search for the mutated fragment. Print out the fragments into which the mutated query is divided (`-L`) and convince yourself only one of them has an exact match in S.

Problem 238 We have already seen global and local alignments. What type of alignment is generated by `kerror`?

Problem 239 Next, we try to locate the *Adh/Adh-dup* locus in the genome of *Drosophila melanogaster*. The genome of *D. melanogaster* consists of three autosomes, two sex chromosomes, and the mitochondrial genome. Their sequences are located in the following files:

Chromosome	Type	File
Left arm of chromosome 2	Autosome	`dmChr2L.fasta`
Right arm of chromosome 2	Autosome	`dmChr2R.fasta`
Left arm of chromosome 3	Autosome	`dmChr3L.fasta`
Right arm of chromosome 3	Autosome	`dmChr3R.fasta`
Chromosome 4	Autosome	`dmChr4.fasta`
Mitochondrial genome	Mitochondrial	`dmChrMt.fasta`
X chromosome	Sex chromosome	`dmChrX.fasta`
Y chromosome	Sex chromosome	`dmChrY.fasta`

Copy the genome files to your working directory and determine the length of each of these DNA sequences (`cchar`). How long is the genome of *D. melanogaster* in total?

Problem 240 The file `hamlet.fasta` contains Shakespeare's *Hamlet* in FASTA format. Copy it to your working directory and take a look at it (`less`) to find the opening sentence (Who's there?). How much longer is the genome of *D. melanogaster* than the tragedy?

Problem 241 Given that the genome files of *D. melanogaster* are large, we can save disk space by deleting them from our working directory

```
rm dmChr*.fasta
```

and creating the corresponding symbolic links:

```
ln -s ../Data/dmChr*.fasta .
```

Does the link have the same content as the original file (`diff`)? What is the size of a link compared to that of the original file (`ls -l`)?

Problem 242 Find *Adh/Adh-dup* in the genome of *D. melanogaster* using `kerror`. For this purpose, write a script that goes over the chromosomes and sets the number of errors

$$k = 1, 2, 5, 10, 20, 50, 100, 200, 500.$$

Where is *Adh/Adh-dup* located, and how many errors does the final alignment contain?

Problem 243 What is the number of errors per site at the *Adh/Adh-dup* locus?

Problem 244 Take another look at the *Adh*/*Adh-dup* alignment returned by `kerror`. Where are most of the errors located?

Problem 245 Instead of calculating the number of errors per site from a global/local alignment, calculate the number of mismatches per site from a local/local alignment. For this purpose, cut out the target region on chromosome `2L` and aligning it to `dmAdhAdhdup.fasta` using `lal`.

Problem 246 Repeat the `kerror` search for *Adh*/*Adh-dup* with the minimum k possible. Observe the run time printed by the program. How is this distributed between exact matching and checking through dynamic programming?

Problem 247 Align `dmGenomic.fasta` with its homologue from *D. guanche* using `gal`. What would k need to be for locating the *D. guanche Adh*/*Adh-dup* in the *D. melanogaster* genome? Is that feasible?

4.2 Fast Local Alignment

Homology between sequences is best determined locally: Even if homology were, in fact, global—or global/local as in k-error alignment—a local alignment would detect these configurations if they maximized the score. As we have seen for k-error alignment, local alignment is sped up by mixing exact and inexact matching. The first step in both methods, is therefore, to divide the query into shorter segments that are to be matched exactly. However, instead of using contiguous segments of variable length as in k-error alignment, in local alignment the segments, which are usually called "words", overlap and have a fixed length, say 11 bp (Fig. 4.2a). These words are then searched for in the subject sequence, which typically consists of an entire database of sequences (Fig. 4.2b). Wherever a hit is found, it is extended to the left and right until the score cannot be improved any further (Fig. 4.2c). In other words, the sketched algorithm returns *ungapped* local alignments. This was implemented in the first version of BLAST from 1990 [6]. Modern versions return gapped alignments [7], but as we shall see below, the ungapped algorithm is already quite effective.

Apart from its clever algorithm, BLAST is fast, because it incorporates a formula for calculating the significance of an alignment. The alternative to using the formula would be to compute P-values from simulations. As we show in this section, calculating P-values by simulation is much slower than by formula. Moreover, an important lesson to take away from this set of Problems is that small adjustments in parameters can have large effects on the alignments returned.

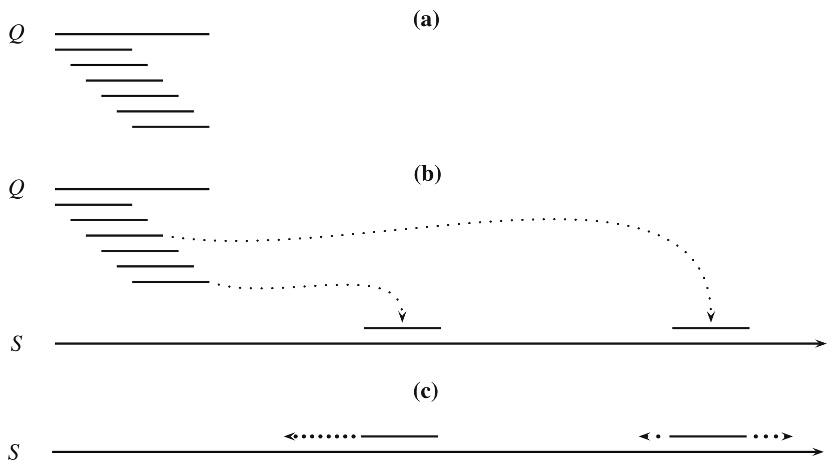


Fig. 4.2 BLAST algorithm. Preprocess the query sequence, Q, into overlapping words (**a**); search the words in the subject, S, (**b**); extend matches until the score cannot be improved any further (**c**)

New Concepts

Name	Comment
fast local alignment	ungapped or gapped version
ungapped alignment	alignment without dynamic programming

New Programs

Name	Source	Help
`blastn`	book website	`blastn -h`
`sblast`	book website	`sblast -h`

4.2.1 Simple BLAST

Problem 248 Create the directory `FastLocalAlignment` and change into it. Copy or link `dmAdhAdhdup.fasta` into your working directory. Cut out positions 3101-3200 from `dmAdhAdhdup.fasta`, save the fragment in `dmAdhFrag.fasta`, and align it to the original sequence using `sblast`. Output the word list generated by `sblast`. How many words does it contain? How long are these words?

Problem 249 Our query has length 100. What happens if it contains ten equally spaced mutations, for example at positions 1, 12,..., 100? Check your answer by writing a script that drives `mutator` to repeatedly mutate the fragment at the desired positions. Make sure the script is applied to a copy of the original fragment called, say, `dmAdhFrag2.fasta`, rather than to the original.

Problem 250 The word length applied by `sblast` can be set by the user. What is the range of values compatible with finding the mutated fragment `dmAdhFrag2.fasta`?

Problem 251 Fast alignment algorithms like BLAST are called "heuristic" in contrast to the "optimal", but slow, algorithms based on the dynamic programming. Can `lal`, which implements an optimal algorithm, align `dmAdhFrag2.fasta`?

Problem 252 Natural mutations are randomly distributed rather than equally spaced. Use `mutator` with mutation rates of 1, 2, 5, 10, 20, and 50 % to generate mutated versions of `dmAdhFrag.fasta` and search for them in `dmAdhAdhdup.fasta` with `sblast`. Iterate 100 times per mutation rate and plot the number of alignments found as a function of mutation rate. Make sure that only one alignment is counted per run. Hint: `grep -A 1` includes the line *after* the match in the output.

Problem 253 Repeat the mutation analysis, only this time set the `sblast` threshold for acceptance of an alignment from its default value of 50 to 25 (`-t`). Plot the results for both thresholds in a single graph.

Problem 254 Use `sblast` to find the best local alignment between `dmAdhAdhdup.fasta` and `dgAdhAdhdup.fasta`. Then use `lal` to find the optimal local alignment. The programs `sblast` and `lal` use the same score scheme—which one finds the better alignment?

Problem 255 By default, `sblast` extends an alignment for up to 20 steps without improving the score before it gives up. What happens if you increase that number to 40 (`-s`) and rerun the the comparison between *D. melanogaster* and *D. guanche*?

Problem 256 Write a script called `driveSblastDm.sh` to drive the comparison between `dmAdhAdhdup.fasta` and all chromosomes of *D. melanogaster* using `sblast`. Hint: There is a special notation for iterating over all command line arguments; to test this, write the script `example.sh`

```
for a in $@
do
   echo ${a}
done
```

and run it as

```
bash example.sh *
```

What is the interval occupied by *Adh*/*Adh-dup*?

Problem 257 Use `time` to compare the run time of `sblast` and `kerror` when searching for `dmAdhAdhdup.fasta` in `dmChr2L.fasta`. Can you explain the run time difference between `sblast` and `kerror`?

Problem 258 Recall from Problem 247 that `kerror` could not locate `dgAdhAdhdup.fasta` in the genome of *D. melanogaster*. Try again with `sblast`. What are the coordinates of the homologous region?

4.2.2 Modern BLAST

Problem 259 Align the artificially mutated sequence `dmAdhFrag2.fasta` to `dmAdhAdhdup.fasta` using `blastn` with default parameters. Do you get a hit? What happens if you adjust the word size option (`-word_size`)?

Problem 260 What is the default word size of `blastn`? To answer this question, again start from the exact match in `dmAdhFrag.fasta`, copy it to `dmAdhFrag2.fasta`, and mutate it using mutate.sh. But this time vary the step length between mutations until you find the smallest step length that gives a hit. To make this more convenient, copy `mutate.sh` to `mutate2.sh` and change its first line from

```
for i in $(seq 1 11 100)
```

to

```
for i in $(seq 1 $1 100)
```

The `$1` refers to the first argument on the command line, which means `mutate2.sh` can now be run as, say,

```
bash mutate2.sh 50
```

where 50 is the distance between mutations. In addition, insert

```
cp dmAdhFrag.fasta dmAdhFrag2.fasta
```

as the first line of `mutate2.sh`. To minimize the number of trials, we use a method called "binary search": Start with the smallest word size, 1, which gives no hit, and the largest, 100, which does. This establishes that the sought word length lies in the interval (1, 100). The trick is now, to repeatedly halve this interval. If the new midpoint results in a match, it becomes the right border of the interval to be halved next; conversely, if the new midpoint results in no match, it becomes the left border.

Problem 261 By default, `blastn` is run in a low-sensitivity mode called "megablast". This can be switched to a high-sensitivity mode by using

```
-task blastn
```

Repeat the binary search from before to find the default `-word_size` in this mode.

Problem 262 Compare the sensitivity of `blastn` in megablast and blastn mode by plotting the % alignments detected as a function of mutation rate, as in Problem 252. Again, start from the exact match in `dmAdhFrag.fasta` and mutate it using `mutator`. Use the tabular output format for this, `-outfmt 7`.

Problem 263 Regenerate `dmAdhFrag2.fasta` with mutation distance 11:

```
bash mutate2.sh 11
```

Table 4.1 Default score schemes of the local alignment programs used in this section

Parameter	`sblast`	`blastn`	`lal`
Match	1	2	1
Mismatch	−3	−3	−3
Gap opening	na	−5	−5
Gap extension	na	−2	−2

and run `blastn` in blastn mode and with appropriate word size like in Problem 259:

```
blastn -task blastn -word_size 10 -query dmAdhFrag2.
    fasta -subjectdmAdhAdhdup.fasta
```

to get the score line

```
Score = 141 bits (156), Expect = 8e-38
```

of which the simplest statistic is the raw score in round brackets, 156. It is computed using the score scheme printed at the end of the BLAST output:

```
Matrix: blastn matrix 2 -3
Gap Penalties: Existence: 5, Extension: 2
```

Under the conventions we have used so far, this means match is 2, mismatch −3, gap opening −5, and gap extension −2. Apart from the match score, these values are identical to those used by `lal`. This is shown in Table 4.1, which summarizes the score schemes for `sblast`, `blastn`, and `lal`. Repeat the `blastn` run with `lal` adjusted such that it returns the same raw score as `blastn`.

Problem 264 The score line begins with the bit score. This is intended to be more independent of the score scheme than the raw score. What happens to these two scores when you set match to 1 instead of the default 2 (`-reward`)?

Problem 265 Use `blastn` to align the *Adh*/*Adh-dup* region from *D. melanogaster* and *D. guanche*. Can `lal` find a better alignment under the same score scheme?

Problem 266 The amount of searching `blastn` does is determined among other things by the heuristic parameter `-xdrop_gap_final`. Its default is 100, higher values generally lead to a more thorough search. Try to find an `-xdrop_gap_final` value that returns the same alignment as `lal`.

Problem 267 We have tuned the parameter `-xdrop_gap_final` to get the same alignment as `lal`. Is it theoretically possible to find a better alignment with `blastn`, for example, by setting a different value for `-xdrop_gap_final`, without changing the score scheme?

Problem 268 Cut out the interval 3101–3200 from the *D. guanche Adh*/*Adh-dup* and save it in `dgAdhFrag.fasta`. Align the fragment with `dmAdhAdhdup.fasta` using `blastn`. Compare the `blastn` result with that obtained by `lal -A 2`. If the two results disagree, try reducing the `-word_size` from its default value of 11 to make the results agree.

Problem 269 The score line of our current BLAST command is

```
Score = 24.7 bits (26), Expect = 0.015
```

We have so far explored the bit score and the raw score. The `Expect` value means that 0.015 alignments between shuffled versions of the query and the subject are expected to have a score of 24.7 or greater. This value is called the expectation value or E-value. It is related to the significance from statistics through

$$P = 1 - e^{-E},$$

which implies $P \leq E$. What is P given $E = 0.015$?

Problem 270 The probability of finding an alignment with score, S, greater or equal to some given score, x, is [6]

$$P(S \geq x) = 1 - e^{-y},$$

where $y = Kse^{-\lambda x}$, s is the "effective" search space, K a correction factor, and λ incorporates the score matrix. These three parameters are quoted in the footer of the `blastn` output. Use their "gapped" version to compute $P(S \geq x)$ for our alignment.

Problem 271 To further explore the E-value computed by `blastn`, and hence the significance of the alignment, we test the null hypothesis that the observed score is due to chance through simulation rather than relying on the theory used by `blastn`. For this purpose, we compute the distribution of scores between random versions of `dmAdhAdhdup.fasta` and the original `dgAdhFrag.fasta`. Use `lal` with `blastn` parameters. Start with a small number of iterations until the simulation works, then carry out a long run with 1000 iterations and store the scores. How long does this take?

Problem 272 Write an AWK script called `count.awk` to compute the frequency of each distinct score. Plot this frequency as a function of the score.

Problem 273 The significance of the original score of 26, that is, its P-value, is the frequency, with which scores $\geq$ 26 appear among the random scores in `simPval.dat`. What is the simulated P-value of the original score? Compare your simulated result to the theoretical P-value implied by the BLAST output.

Problem 274 As a final step in our exploration of alignment statistics, we compare the theoretical score distribution to that generated by simulation in Problem 272. The probability of observing a score S is

$$P(S) = \lambda e^{w}, \tag{4.1}$$

where $w = (\mu - S)\lambda - e^{(\mu - S)\lambda}$ and $\mu = \ln(Ks)/\lambda$. The program `gnuplot` allows the definition of variables and functions, which can then be plotted; for example:

```
a=10
f(x)=a*x
plot f(x)
```

Use this feature to overlay the histogram from Problem 272 with the theoretical curve given by Eq. (4.1).

Problem 275 Write a script called `blast.sh` to search `dmAdhAdhdup.fasta` across all chromosomes of *D. melanogaster*. To minimize spurious results, use a minimum E-value of 10^{-20} by setting

```
-evalue 1e-20
```

Reduce clutter by switching to tabular format (`-outfmt 7`). How long does the search take?

Problem 276 Searches can be sped up by turning the subject sequences into a BLAST database:

```
cat dmChr*.fasta |
makeblastdb -dbtype nucl -title dmDb -out dmDb
```

How long does this take? What is the run time for searching `dmAdhAdhdup.fasta` in the database?

Problem 277 What happens to the run time when using the default megablast mode and the database? Does the mode affect the result?

Problem 278 Repeat the search with `dgAdhAdhdup.fasta`. Again compare the results obtained in blastn and megablast mode.

4.3 Shotgun Sequencing

The quest for ever faster alignment methods is driven by the ease with which genomes can be sequenced these days. When sequencing a DNA molecule, biologists initially need to know the sequence of about 20 nucleotides. This allows them to synthesize an oligonucleotide complementary to the known sequence. The oligo is then used to start, or *prime*, a sequencing reaction. Any such reaction reveals the identity of at most a few hundred nucleotides. A naïve way to sequence longer molecules would, therefore, be to obtain the first sequencing result, design a new primer to the 3' end of the sequence just determined, and repeat the cycle of sequencing and primer design. However, this sequential walking along a chromosome is slow. It is much more efficient to parallelize the procedure: Fragment a large number of copies of the template molecule through, for example, sonication. Then insert the random fragments into a piece of DNA with known sequence. The sequencing reaction can now be primed using always the same oligos complementary to the known flanking DNA. This "shotgun sequencing" was invented by the English biochemist Fred Sanger in

1982 [43] and yields random sequences, or *reads*, which need to be assembled into the template sequence. Assembly is carried out by programs that look for overlaps between a potentially very large number of reads.

In the following section, we start with optimal overlap alignments and end with a fast method for assembling the genome of *Mycoplasma genitalium* from simulated shotgun reads.

New Concepts

Name	Comment
overlap alignment	sequence assembly
shotgun sequencing	rapid sequencing method

New Programs

Name	Source	Help
`oal`	book website	`oal -h`
`sequencer`	book website	`sequencer -h`
`velvetg`	book website	`velvetg`
`velveth`	book website	`velveth`

Problem 279 The two sequences, S_1 = `ACCGTTC` and S_2 = `GTTCAGTA` overlap. Write down their alignment to show this.

Problem 280 Create the directory `Shotgun`, change into it and write S_1 and S_2 into the FASTA files `s1.fasta` and `s2.fasta`, respectively. Then use the program `oal` to compute the overlap alignment between S_1 and S_2.

Problem 281 Sequencing reads can originate from the forward or the reverse template strand. How many comparisons are necessary between two sequences to find the best overlap if you take strand into account?

Problem 282 The files `f1.fasta`, `f2.fasta`, and `f3.fasta` each contain a fragment of the *M. genitalium* genome. Copy them from the `Data` directory to your working directory. Think of these fragments as sequencing reads from a shotgun experiment. To find the overlaps between them, we need to compare each fragment to the forward and reverse strand of the other two fragments. If we denote the forward and reverse strands of the i-th fragment as f_i^{f} and f_i^{r}, the following combinations of fragments need to be checked:

	f_1^{f}	f_1^{r}	f_2^{f}	f_2^{r}	f_3^{f}	f_3^{r}
f_1^{f}			●	●	●	●
f_1^{r}						
f_2^{f}					●	●
f_2^{r}						
f_3^{f}						
f_3^{r}						

Think of the names along the first column as query, the names along the first row as subject. Write down the score for each comparison. What are the two most substantial overlaps and what does this say about the relationship between the three fragments? Hint: A read can be reverse complemented using `revComp`. Also, start by creating files with names like `f1f.fasta` and `f1r.fasta`.

Problem 283 Look at the overlap alignments of the two most substantial overlaps just discovered. Draw by hand a figure with alignment coordinates to show how the reads overlap. Then calculate the length of the genomic sequence from which the reads were taken.

Problem 284 Every sequencing project starts with the isolation of the template DNA molecule. We simulate this step by generating a random sequence. To give us an idea of how long a realistic genome might be, use `cchar` to compute the length of the genome of *M. genitalium* contained in the file `mgGenome.fasta`. Also, what is the fraction of `G` and `C` nucleotides in *M. genitalium*, that is, its GC content?

Problem 285 Use `ranseq` to generate a random genome with the same length and GC content as *M. genitalium*, and store it in `ranGenome.fasta`. To obtain exactly the same genome as we did, initialize the random number generator in `ranseq` with 35. Check your result with `cchar`.

Problem 286 What is the difference between `ranseq` and `randomizeSeq`?

Problem 287 A shotgun sequencing experiment yields a—usually large—number of random reads comprising a total of s nucleotides. Such an experiment is characterized by the coverage, $c = s/L$, where L is the length of the molecule sequenced. How many bases would you get if the sequencing facility at your institution sequenced `ranGenome.fasta` to a coverage of 10?

Problem 288 We now study the relationship between the coverage, c, and the probability of sequencing a particular nucleotide. If we picture "sequencing" as randomly drawing a single nucleotide from the genome, the probability of getting a particular one is $1/L$ and the probability of not getting it is $1 - 1/L$. The probability of not sequencing a nucleotide in s trials is

$$P_0 = \left(1 - \frac{1}{L}\right)^s .$$

To simplify this, we first rewrite

$$\left(1-\frac{1}{L}\right)^{s}=e^{s\ln(1-\frac{1}{L})},$$

and use the approximation $\ln(1+x)\approx x$ to get

$$P_0 \approx e^{-s/L} = e^{-c}.$$

How many nucleotides are expected to be left unsequencedif the random genome is shotgunned to a coverage of 10?

Problem 289 What is the theoretical coverage necessary to achieve a combined gap length of 1?

Problem 290 What is the theoretical coverage necessary to achieve a combined gap length of 0?

Problem 291 Use `sequencer` to sequence your random genome to the coverage that would yield an expected gap length of 1. Store the reads in `reads.fasta`. How many reads did you generate? How many nucleotides?

Problem 292 We use `velvet` for assembly. `Velvet` first "hashes" the reads, which means indexing overlapping read fragments of some length. These fragments are then used as seeds for inexact matches between reads. Use `velveth` to hash the short reads to a length of 21 and `Assem` as the name of your assembly directory.

Problem 293 Assemble the hashed reads using `velvetg`. The only option you need to set at this stage is the expected coverage, `-exp_cov`. `velvetg` assembles the reads into contiguous sequences, or *contigs*. How many contigs do you get and how many nucleotides do they comprise?

Problem 294 So far we have simulated sequencing projects by picking random reads. In real sequencing projects, random fragments of mean length, say, 500 bp are picked and sequenced from both sides. This approach is called *paired-end sequencing*. The information about read pairing is passed on to the assembly program. Repeat the sequencing experiment, but this time generate paired-end reads in `sequencer` and use the corresponding option in `velveth`. In `velvetg` set the "insert length", that is, the length of the fragment sequenced from both ends, to 500. How many contigs and nucleotides do you get?

Problem 295 You have sequenced with the default error rate of 0.1%, which means the sequencer mis-calls one in 1000 nucleotides. What happens if you eliminate errors altogether?

Problem 296 Increase the error rate to 1%. How many contigs and nucleotides do you get?

Problem 297 Sequence and assemble the genome of *M. genitalium*. Generate single-end reads in `sequencer` with default error and the coverage for an expected gap length of 1 calculated in Problem 289. How many nucleotides do you get in how many contigs?

Problem 298 So far, we have assessed the quality of alignments only by counting the contigs and the nucleotides they contain. However, a popular measure of assembly quality is the N_{50}. This is related to the median contig length, which in turn is similar to the mean contig length. So before, we define the N_{50}, we remind ourselves of median and mean. Consider five toy contigs with lengths $\mathscr{L} = \{2, 2, 3, 4, 5\}$. What is the median and the mean contig length?

Problem 299 The N_{50} is the length of the shortest contig contained in the set of longest contigs that comprise at least 50% of assembled nucleotides. What is the N_{50} of our toy contigs, $\mathscr{L}$?

Problem 300 On its last output line, `velvetg` also prints the N_{50} (`n50`), because the larger the N_{50}, the better the assembly. What is the N_{50} of the assembly you have just computed?

Problem 301 We now write a pipeline to compute the N_{50} for the set of contigs assembled in the previous Problem. First, we write a pipeline for extracting and sorting the contig lengths. It should, executed from left to right,

- report the length of each contig (`cchar -s`);
- extract the lines that contain the contig length (`grep`);
- print only the contig length (`awk`);
- sort the lengths in descending order (`sort`).

What is the median contig length (`tail` & `head`)?

Problem 302 Now we are in a position to write an AWK program for computing the N_{50} from ordered contig lengths. Call this program `n50.awk`; it

- stores the lengths of all contigs;
- extracts the total length of all contigs;
- sums the contig lengths, and once that sum is greater than half the total length, prints the current contig length. Use a `while` loop for this step.

Problem 303 Repeat the sequencing and assembly with paired-end reads and compute the N_{50}. Save this final assembly to `mgAssembly.fasta`.

4.4 Fast Global Alignment

In Sect. 4.3 we assessed the quality of our genome assembly using such metrics as the number of contigs, the number of bases assembled, and the N_{50}. However, the ultimate quality check is to align the assembly and the template. Since these two data sets comprise over 500 kb, the corresponding alignment matrix would be huge. Instead, we can apply fast, "heuristic" alignment methods that are based on a clever mix of exact matching and optimal alignment. In this section, we use the package MUMmer [15, 29] for fast global alignment. Specifically, we assess the assembly of the *M. genitalium* genome and to compare two strains of *Escherichia coli*. The core program in the MUMmer package, `mummer`, is based on a suffix tree. In Sect. 3.2 the suffix tree was constructed from a single sequence. This can be generalized to an arbitrary number of sequences by adding the suffixes of them all. However, we now need to keep track of where a suffix came from and hence label the leaves with a pair of numbers: (string, suffix). For example, Fig. 4.3a shows the suffix tree for `ADAM` already familiar from Fig. 3.5. We first add a sentinel character (Fig. 4.3b) and then augment the leaf labels by information on string of origin (Fig. 4.3c). Finally, we insert the suffixes of `DAD` (Fig. 4.3d). A leaf can now have more than one label; for example (1, 5) and (2, 4) for the sentinal suffix, `$`. In this section, we first learn how generalized suffix trees can be used to find repeats between genomes. Then, we use MUMmer to compare bacterial genomes.

	1	2	3	4	5
S_1	A	D	A	M	$
S_2	D	A	D	$	

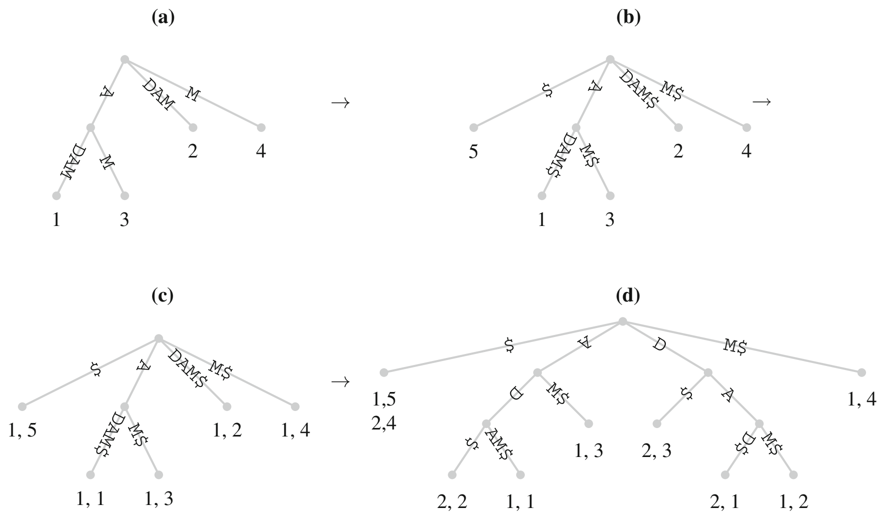


Fig. 4.3 Construction of the generalized suffix tree for `ADAM` and `DAD`. (**a**): `ADAM` alone; (**b**): `ADAM` plus sentinel (`$`); (**c**): `ADAM$` with leaf labels of the form (sequence, suffix); (**d**): `ADAM$` and `DAD$` in one suffix tree

New Concept

Name	Comment
generalized suffix tree	suffix tree for ≥ 1 sequences

New Programs

Name	Source	Help
`drawStrees`	book website	`drawStrees -h`
`mummer`	book website	`mummer -h`
`nucmer`	book website	`nucmer -h`
`show-snps`	book website	`show-snps -h`

Problem 304 Draw by hand the generalized suffix tree of $S_1 =$ `AAGCG` and $S_2 =$ `AGT`.

Problem 305 Use the program `drawStrees` to draw the generalized suffix tree of S_1 and S_2. `drawStrees` generates a LaTeX file as output. Convert this to a viewable postscript file as described in Problem 185.

Make sure that in the FASTA input for `drawStrees` the second sequence is terminated by a carriage return.

Problem 306 Label by hand each internal node in your suffix tree with the length of the path label, the string depth. By searching for the node with the greatest string depth you can find the longest repeat in the input data. What is the longest repeat between S_1 and S_2?

Problem 307 Determine the length and composition of the assembly in `mgAssembly.fasta` generated in Problem 303 and compare this to the length of the original genome.

Problem 308 Before we compare `mgAssembly.fasta` to `mgGenome.fasta`, we carry out a few simple experiments with `mummer` to help interpret its output. Use `ranseq` to generate a random 1 kb sequence and save it in `s1.fasta`. Then use `cutSeq` to cut the first and last 100 bp from `s1.fasta`, splice them together, and save the resulting single 200 bp sequence to `s2.fasta`. Compare `s1.fasta` and `s2.fasta` using `mummer`. Can you interpret the output?

Problem 309 In its simplest form, the output of `mummer` is a dot plot, albeit a very efficiently computed dot plot. Use the program `mum2plot.awk` to convert the `mummer` output to `gnuplot` input.

```
mummer s1.fasta s2.fasta | awk -f mum2plot.awk |
    gnuplot ...
```

In the resulting plot, which sequence is written along the x-axis, which along the y-axis?

Problem 310 Reverse-complement `s2.fasta` using `revComp`, save it to `s3.fasta`, and align it again to `s1.fasta`; only this time we require that `mummer` returns matches on both the forward and reverse strand (`-b`) and that these matches are reported relative to the original sequence, rather than the reverse complement (`-c`). Plot the result as before.

Problem 311 As a final preparatory step, write the contents of `s2.fasta` and `s3.fasta` to `s4.fasta` and compare it to `s1.fasta`. Before carrying out the computation, make a sketch of the plot you expect so see.

Problem 312 When comparing the output of `mgAssembly.fasta` to `mgGenome.fasta`, it is easier to interpret the plot if the assembly consists of a single sequence. To concatenate the contigs into a single sequence, first create a new file with a FASTA header

```
echo '>Assembly' > mgAssemblyS.fasta
```

Then append the contigs with their headers removed:

```
sed '/^>/d' mgAssembly.fasta >> mgAssemblyS.fasta
```

Compare `mgGenome.fasta` as reference to `mgAssemblyS.fasta` as query using `mummer`. As before, it should compute matches on both the forward and the reverse strand, and report all query positions with respect to its forward strand. Hint: `mummer -help`.

Problem 313 Our next goal is to compare the genomes of two assembled *E. coli* strains contained in `ecoliK12.fasta` and `ecoliO157H7.fasta`. How long are these two genomes?

Problem 314 Use `mummer` to draw the dot plot for K12 and O15H7 with K12 as reference. What do you notice about the two sequences?

Problem 315 Alignments are used to compare sequences at nucleotide resolution, something that cannot be done with dot plots. Resolution at the nucleotide level can, in turn, be used to estimate the number of mutations, or single nucleotide polymorphisms (SNPs), that separate two genomes. The program `nucmer`, which is part of the mummer package, aligns genomes ready for SNP discovery by `show-snps`:

```
nucmer -p nucmer ecoliK12.fasta ecoliO157H7.fasta
show-snps nucmer.delta > nucmer.snps
```

What is the number of SNPs per nucleotide between K12 and O157H7?

Problem 316 Given that the mutation rate in *E. coli* is 2.2×10^{-10} per generation [31], what is the divergence time between K12 and O157H7 in generations?

Problem 317 The doubling times of *E. coli* range from 30 to 90 min [39]. What is the corresponding range in years to the most recent common ancestor for K12 and O157H7?

4.5 Read Mapping

Sequencing reads are among the most common data in Molecular Biology. We have already seen one typical application of sequencing, shotgun sequencing, where reads are assembled into the template sequence. In contrast to this *de novo* sequencing, re-sequencing is carried out in organisms with established reference genome sequences. In this situations, reads are mapped to the reference, often for discovering genetic variants. In this section, we explore the time requirements of read mapping, and how read alignments can be manipulated.

New Concept

Name	Comment
read mapping	variant discovery

New Programs

Name	Source	Help
`bwa`	book website	`bwa`
`samtools`	book website	samtools

Problem 318 Create the directory `ReadMapping` for this session and change into it. Use the program `sequencer` to simulate the sequencing of `dmChr2L.fasta` with default parameters, except for coverage, which should be 15 instead of 1. In addition, you can synchronize your results with ours by using 10 as the seed for the random number generator (`-s`). Save the reads in `reads.fasta`. How many reads were generated (`grep`)? Do they represent the desired coverage (`cchar`)?

Problem 319 The program `sequencer` picks a fragment of mean length 500 bp with standard deviation $\sqrt{500} \approx 22.4$ (see `sequencer -h`). From such a fragment it generates a read of length 100 bp. Modern sequencers produce reads of identical lengths and hence, the standard deviation of read length is zero by default. However, occasionally `sequencer` produces a read shorter than the default length. How many of these do you find in your data set? Can you explain how they might occur?

Problem 320 Map the reads using the BLAST-mode blastn-short for mapping short reads (`-task`). But before mapping all reads, write a script for measuring the run times when mapping 1, 2, 5, 10, 20, 50, 100, 200, 500, and 1000 reads. How long would it take to map all reads?

Problem 321 The program `bwa` is a read mapper based on the Burrows–Wheeler transform, hence the name [33]. It works on an index of the reference sequence, which is computed with the command

```
bwa index -p dmChr2L dmChr2L.fasta
```

How long does index computation take? The program reports the main computational steps. Do you recognize any of the operations being described?

Table 4.2 Mandatory fields in SAM file

Col.	Meaning
1	Query
2	Comment flag; 0: none, 4: unmapped, 16: reverse-complement
3	Subject
4	Position
5	Mapping quality
6	Match string; M: match, D: deletion, I: insertion, S: soft clipping from read
7	Name of read mate
8	Position of read mate
9	Template length
10	Read sequence
11	Base quality

Problem 322 Like with BLAST in Problem 320, align 1, 2, 5, 10, 20, 50, 100, 200, 500, and 1000 reads and redirect the output to the null device. But unlike the BLAST run, these are *thousands*, which requires changing the assignment to `x` in line 4 of `runBlast.sh`. Call the new script `runBwa.sh`. Plot the run times and estimate how long it would take to align all reads.

Problem 323 Align all reads and save the result in `reads.sam`. How long does this take? Compare the observed run time to the estimate from Problem 322.

Problem 324 Take a look at `reads.sam` (`head`). Lines starting with `@` are header lines. The body of the file contains eleven mandatory columns, which are listed in Table 4.2. Column 6 contains a description of the alignment like 100M, which means 100 matches, in other words, an exact match. Find such an exact match and use `keywordMatcher` to locate it in `dmChr2L.fasta`. How are reverse-complemented reads represented in `reads.sam`?

Problem 325 The first things we might want to do with a SAM file is to look at the alignments. This is done using the program `samtools`. To prepare `reads.sam` for viewing, convert it first to its binary format and sort it:

```
samtools view -b reads.sam | samtools sort > reads.bam
```

These commands can be sped up by assigning more than one thread to each process (`-@`). Next, the BAM file is indexed

```
samtools index reads.bam
```

and can be viewed

```
samtools tview -reference dmChr2L.fasta reads.bam
```

Can you explain the commas and dots you see?

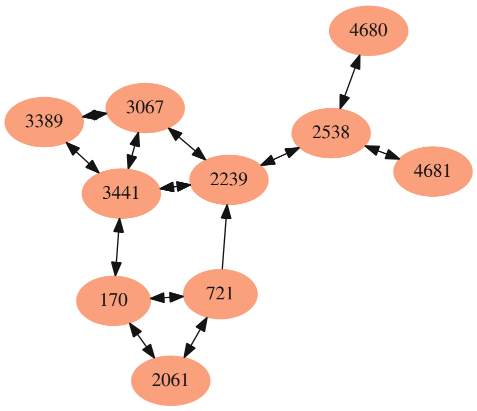


Fig. 4.4 Example of a protein family defined by BLAST hits. Nodes represent proteins, edges are either reciprocal hits of the form query $\leftrightarrow$ subject, or unidirectional hits between query $\rightarrow$ subject

Problem 326 The help menu of `tview` is switched on by pressing "?". What is the command for visiting a particular position in the alignment?

Problem 327 Positions in an alignment are denoted by

```
sequenceName:position
```

The names of the template sequences mapped to are listed in the header of a SAM/BAM file:

```
samtools view -H reads.bam
```

where `SN` in the output refers to "sequence name". Go to position 2000. Given a position annotation like

```
2001
ATTC
```

which digit refers to the actual position?

4.6 Clustering Protein Sequences

In this section, we use protein BLAST to find out whether the genome of *M. genitalium* contains sets of genes with similar function. We do this by searching for sets of genes with similar sequences and assume tacitly that sequence similarity implies similar function. Such sets of similar genes are called "gene families". Gene families arise in the course of evolution by gene duplication [35], as seen with *Adh*/*Adh-dup* in *Drosophila*. A gene family might look like the graph in Fig. 4.4, where the nodes represent proteins and the edges BLAST hits, i.e., the arrows pointing from query to subject.

In homologous pairs, either of the two proteins involved can serve as query/subject. If there is a hit with both labelings, it is called "reciprocal" and is denoted as query $\leftrightarrow$ subject. Otherwise we just have a hit in one direction, query $\rightarrow$ subject. Reciprocal hits are the norm. The one exception in Fig. 4.4 is the pair 721 $\rightarrow$ 2239, where the comparison is only significant if 721 is query, not if 2239 is query.

Gene families are common in the large genomes of, for example, mammals. The human genome encodes approximately 23,000 genes of which hundreds belong to several families of olfactory receptors [11], a reflection of the importance of the sense of smell in terrestrial mammals. However, *M. genitalium* has one of the smallest genomes of any free living organism with less than 500 protein-coding genes. The question we investigate in this section, is therefore, whether the genome of *M. genitalium* is too small to accommodate gene families.

New Concept

Name	Comment
protein family	set of proteins with similar sequences and functions

New Programs

Name	Source	Help
`blast2dot.awk`	book website	`blast2dot.awk -v h=1`
`blastp`	book website	`blastp -help`
`circo` (GraphViz)	package manager	`man circo`
`getSeq`	book website	`getSeq -h`
`neato` (GraphViz)	package manager	`man neato`
`ranDot.awk`	book website	`ranDot.awk -v h=1`

Problem 328 How many edges could in theory be drawn between the nodes (proteins) in Fig. 4.4? What proportion of these is actually found?

Problem 329 Make a directory for this session,

```
FastLocalAlignmentProt
```

change into it, and link the file `mgProteome.fasta`, which contains all proteins of *M. genitalium*. How many proteins are there?

Problem 330 In order to find protein families in *M. genitalium*, we shall carry out an all-against-all comparison of the proteins. How many comparisons does this comprise? Test your prediction by producing five identical DNA sequences using `ranseq` with the same seed for the random number generator, and running `blastn` on that toy data set.

Problem 331 Carry out an "all-against-all" comparison on `mgProteome.fasta` with $E = 10^{-5}$. Save the result in tabular format in the file `mgProteome.blast`. Since we are only interested in the closest homologue of each sequence, restrict the output to at most one hit per query:

```
-max_hsps 1
```

How long does it take `blastp` to carry out the approximately 230,000 comparisons (`time`)? Take a look at the BLAST result.

Problem 332 Convert `mgProteome.fasta` to the BLAST database `mgProteome` (`makeblastdb`). How long does this take? How long does the all-against-all comparison take using the database? Again, save the result in `mgProteome.blast`.

Problem 333 Every accession in `mgProteome.blast` has the redundant prefix `lcl|`. Use `sed` to remove it. Hint: Save the result of `sed` to a temporary file, say `tmp`, before replacing the original with the edited version.

Problem 334 What happens if the input file of a command is also the output, without saving to an intermediary file? For example

```
sed expr mgProteome.blast > mgProteome.blast
```

Try this, but only after making a backup copy of `mgProteome.blast` first.

Problem 335 Look at the first ten lines of `mgProteome.blast`. Can you find a protein with at least one homologue in the proteome (other than itself)? Use `grep` on `mgProteome.fasta` to discover the function of the first such protein in the list and of its homologue.

Problem 336 How many lines are contained in `mgProteome.blast`? Is it sensible to analyze this file manually to find protein families?

Problem 337 What is the protein with the largest number of homologues in the proteome of *M. genitalium* (`sort`, `uniq -c`)?

Problem 338 Write a pipeline that prints out the list of unique MG-numbers of the proteins linked to MG_410 by BLAST hits. Save the names of this protein family in `protFam.txt`.

Problem 339 The program `neato` is used for visualizing large graphs. We plan to use it for visualizing the network of BLAST hits in the proteome of *M. genitalium*. `Neato` takes input written in the dot language. Here is an example:

```
graph G {
   a -- b
   b -- c
   c -- a
}
```

If this is contained in `example1.dot`, `neato` calculates the layout and visualizes it

```
neato -T x11 example1.dot
```

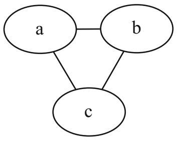
If the x11-terminal is not available on your system, try generating postscript output

```
neato -T ps example1.dot > example1.ps
```

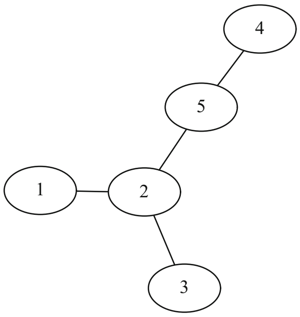
which can then be viewed

```
gv example1.ps &
```

to give



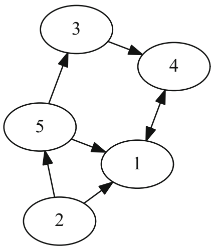
Write a file `example2.dot` that specifies the following graph:



Problem 340 To represent reciprocal and unidirectional BLAST hits, use

```
a -- b [dir=both]     # reciprocal hit
c -- d [dir=forward]  # unidirectional hit
```

which sets the edge attribute `dir`, the default value of which is `none`. Write the dot code for



Problem 341 Use the program `ranDot.awk` to draw a random graph with ten nodes, directed edges, and an edge probability of 0.5:

```
awk -f ranDot.awk -v n=10 -v p=0.5 -v dir=1 > ranDot.
   dot
```

What is the expected number of edges in this random graph? Compute the observed number of edges, counting reciprocal edges double. Compare the expected number of edges to that observed.

Problem 342 Visualize `ranDot.dot` using `neato`. Then visualize it with the program `circo`, which has the same syntax as `neato` and is also part of the Graphviz package. Which graph do you prefer?

Problem 343 We now convert the hits in `mgProteome.blast` to dot code:

```
awk -f blast2dot.awk mgProteome.blast |
sed 's/MG_//g' > mgProteome.dot
```

Notice the `sed` command for removing `MG_` to save space when drawing the nodes. Choose between `neato` and `circo` to draw the graph. What is the size of the largest protein family? Compare it to `protFam.txt`, and if there are any differences, write the new version of the protein family to `protFam2.txt`.

Problem 344 By default, `blast2dot.awk` only includes proteins with at least one BLAST hit in the output. To include proteins without homologues, repeat the computation of dot code including the "singletons"

```
awk -f blast2dot.awk -v singletons=1 mgProteome.blast|
sed's/MG_//g' > mgProteome2.dot
```

What proportion of the proteome are singletons?

Problem 345 What is the function of the members of the extended protein family? Find out by looking up the annotations contained in their FASTA headers using `grep` with extended regular expression syntax, which allows OR. For example,

```
grep -E '(a|b)'
```

reports lines matching `a` or `b`. Hint: Use `protFam2.txt` to automatically construct the list of alternative matches.

Problem 346 To learn more about the protein family just obtained, we search a database of protein motifs and annotations. Prosite is one such database. It consists of two files, `prosite.dat` and `prosite.doc`. The file `prosite.dat` contains motifs, `prosite.doc` annotations. Search `prosite.doc` for our gene family. What is its function?

Problem 347 Why should a bacterium possess multiple ABC transporters?

4.7 Position-Specific Iterated BLAST

For all the alignments computed so far, we used the same score scheme at every position; any mutation between, say, a proline and a histidine had the same score, regardless of whether it occurred in an active site, a transmembrane helix, or an extracellular loop. An alternative to the traditional position-invariant score matrices such as PAM70 and BLOSUM62, are position-specific score matrices; here is an example:

```
        A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V
  1 M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1
  2 E -1  0  0  1 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2
  3 K -1  2  0 -1 -3  1  1 -1 -1 -3 -2  4 -1 -3 -1  0 -1 -3 -2 -2
...
329 N -2  0  6  1 -3  0  0  0  1 -3 -4  0 -2 -3 -2  1  0 -4 -2 -3
```

At every position the query amino acid is listed next to the 20 possible scores. Such a matrix has the advantage of reflecting the local variation in mutation probabilities. Its disadvantage is that every query/subject combination requires a new score matrix. Therefore, position-dependent score matrices cannot be precomputed; they need to be constructed on the fly through iteration. In the first iteration, a conventional score matrix is used to find the closest homologues of the query, which are converted to an initial position-dependent score matrix. This is applied in the second round of sequence comparison. Since searches based on position-dependent score matrices are usually more sensitive than traditional searches, the second iteration often uncovers new homologues. These are used to revise the score matrix before repeating the search. The cycle of searching and matrix revision continues until no new sequences are found. The BLAST program `psiblast`, for Position-Specific Iterated BLAST [7], implements such an iterative search. In this section, we use it to search for further members of the ABC transporter gene family in the proteome of *M. genitalium*.

New Concept

Name	Comment
position-specific iterated BLAST	more sensitive than regular BLAST

New Program

Name	Source	Help
`psiblast`	book website	`psiblast -h`

Problem 348 The program `psiblast` takes as input a query sequence and a database in BLAST format. Set up the directory `PsiBlast` for this session. Then convert `mgProteome.fasta` to the BLAST database `mgProteome`. We previously found that MG_410, together with MG_180 and MG_179, has the largest number of BLAST hits in the proteome of *M. genitalium*. So save MG_410 to `mg_410.fasta` and take it as the starting sequence for our initial `psiblast` run. Leave all options unchanged except for the output format, which should be tabular. Save the output in `mg_410.psi`. How many distinct hits have an E-value less than 10^{-5}?

Problem 349 Repeat the comparison of `mg_410.fasta` to the full proteome with `blastp`. Save the result in `mg_410.bp` and compare it to `mg_410.psi`. Can you find any differences?

Problem 350 Rerun `psiblast` iteratively until convergence by using

```
-num_iterations 0
```

Save the output in `mg_410b.psi`. How many rounds does `psiblast` go through? How many distinct proteins have an E-value of 10^{-5} or smaller in the last round? Save the corresponding proteins in `psiBlastList.txt`. Hint: Each round starts with the best hit, MG_410 to itself.

Problem 351 The list of ABC transporter proteins extracted from the graph of BLAST hits in *M. genitalium* comprised 18 sequences and was recorded in `protFam2.txt`:

```
MG_014 MG_015 MG_042 MG_065 MG_079 MG_080 MG_119 MG_179 MG_180
MG_187 MG_290 MG_303 MG_304 MG_410 MG_421 MG_467 MG_526 MG_390
```

What are the extra three proteins just identified?

Problem 352 What are the annotations in `mgProteome.fasta` of the extra proteins just found?

Problem 353 Next, we investigate the position-specific score matrix used by `psiblast`. To generate it, rerun `psiblast` and save the score matrix in `psiBlast.mat` using

```
-out_ascii_pssm psiBlast.mat
```

Which part of `psiBlast.mat` contains the position-specific score matrix? How many lines does this matrix consist of? Compare the length of the matrix to the length of MG_410.

Problem 354 Scores for a particular amino acid can vary widely along a protein sequence. To illustrate this, first identify the most frequent amino acid in MG_410 (`cchar`). Then use AWK to extract the scores for each position occupied by that amino acid from `psiBlast.mat`. What is the range of its match scores? Compare that to the corresponding match score in BLOSUM62.

4.8 Multiple Sequence Alignment

Up to now we have looked only at alignments between two sequences. But one often needs to align more than two sequences. To do this optimally for n sequences, an n-dimensional dynamic programming matrix is needed. Figure 4.5 illustrates such multidimensional matrices starting from zero dimensions, a mere dot in Fig. 4.5a. By doubling this dot and drawing a connecting edge, a one-dimensional matrix is

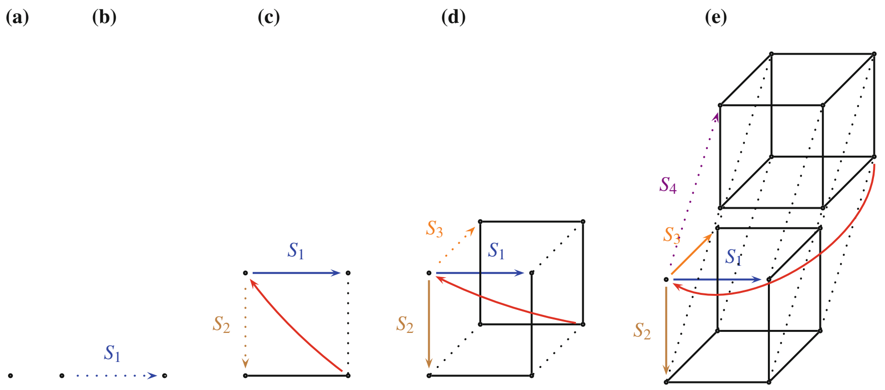


Fig. 4.5 Building multidimensional dynamic programming matrices for optimal multiple sequence alignment. The number of dimensions ranges from zero (**a**) to four (**e**) and corresponds to the number of sequences, S_i, that can be written along its edges and hence aligned. Sequences are indicated by colored arrows labeled S_i. They all start at the same node. The red arc indicates the trace-back

generated, which can accommodate a single sequence, S_1, written from left to right as indicated by the arrow in Fig. 4.5b. In the next doubling step we get the familiar two-dimensional matrix for aligning two sequences, S_1 and S_2, in Fig. 4.5c. The trace-back is indicated by the red arc. Repeat the doubling to get a cube for aligning three sequences (Fig. 4.5d). This can be repeated to create the four-dimensional hyper-cube in Fig. 4.5e.

Clearly, the space and time required for aligning by dynamic programming is multiplicative in the lengths of the input sequences. As a result, there are only few programs implementing optimal multiple sequence alignment [20, 34]. In practice, "shortcuts", or "heuristics", are taken. The most important of these is to reduce a multiple sequence alignment to a set of pairwise comparisons. In this section, we use two versions of this heuristics, query-anchored alignment, and progressive alignment to analyze sets of hemoglobin sequences.

New Concepts

Name	Comment
optimal multiple sequence alignment	generalize optimal pairwise alignment
progressive multiple sequence alignment	heuristic multiple sequence alignment method
query-anchored alignment	heuristic multiple sequence alignment method

New Programs

Name	Source	Help
`clustalw` or `clustal2`	book website	`clustal[w2] -help`
`fasta2tab.awk`	book website	`fasta2tab.awk -v h=1`
`shuffle.awk`	book website	`shuffle.awk -v h=1`

4.8.1 Query-Anchored Alignment

Problem 355 Set up the directory `Msa` for this session and obtain the following four hemoglobin sequences from Uniprot/Swissprot collection of curated protein sequences. This is contained in `uniprot_sprot.fasta`, from where individual sequences can be extracted using `getSeq`:

Accession	Protein
HBA_HUMAN	Human α-hemoglobin
HBA_HORSE	Horse α-hemoglobin
HBB_HUMAN	Human β-hemoglobin
HBB_HORSE	Horse β-hemoglobin

Save the sequences in files called `hbaHuman.fasta`, `hbaHorse.fasta`, etc.

Problem 356 The program `blastp` has an output format for anchoring all subject sequences to the query.

```
-outfmt 2
```

Use this with human α-hemoglobin as query and the other three as subject. Begin by constructing a file containing the subject sequences. The output contains amino acids printed like this

```
\
|
G
```

Can you guess what this means?

Problem 357 Repeat the query-anchored alignment, only this time use human β-hemoglobin as the subject.

4.8.2 Progressive Alignment

Problem 358 Query-anchored alignments have the disadvantage that they depend on the query. The progressive alignment algorithm, which is the most widely used method for computing multiple sequence alignments, avoids this. Given a set of sequences (Fig. 4.6a), progressive alignment starts by computing their pairwise alignments and hence distances (Fig. 4.6b). The distances are summarized as a tree (Fig. 4.6c), which groups the pair with the smallest distance first. At each internal node in the tree two clusters are joined. The tree is traversed from the leaves toward the root, and at the first internal node encountered the sequences of the corresponding leaves, A and D, are aligned (Fig. 4.6d). At the next internal node up, the pair B, C is aligned (Fig. 4.6e). At the root, finally, the two pairwise alignments generated so far, (A, D) and (B, C), are each treated as a unit and aligned with each other in the final pairwise alignment (Fig. 4.6f). In this way, as the algorithm progresses toward the root, only pairwise alignment problems need to be solved, which greatly simplifies the task of aligning multiple sequences. The disadvantage of this procedure is that any gaps introduced early in the algorithm cannot be changed later. This is why the tree is traversed from the leaves up rather than from the root down; the most similar sequences are aligned first, and their alignments contain the fewest of those irrevocable gaps.

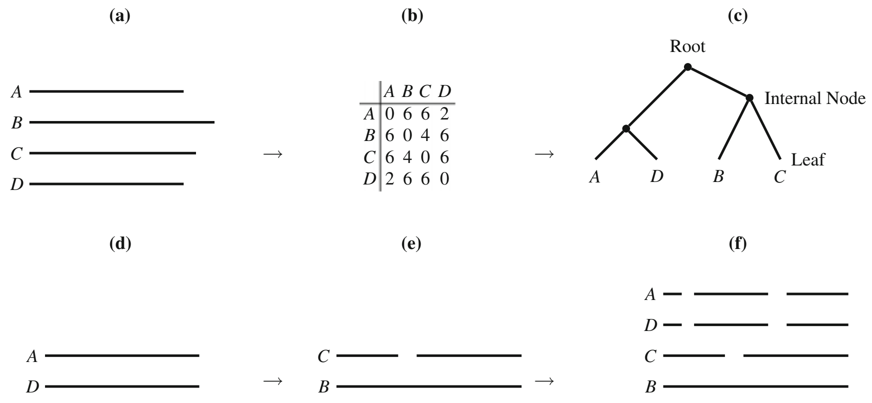
We now recapitulate these steps using our four example sequences. To begin with, compute the $\binom{4}{2} = 6$ pairwise scores using `gal` together with the BLOSUM62 matrix. To convert a given score, s_{ij}, to a distance, d_{ij}, let s_m be the maximum score; then we compute the distance by dividing by the maximum and taking the complement:

$$d_{ij} = 1 - s_{ij}/s_\mathrm{m}.$$

Problem 359 From the distance matrix, construct the order of clustering in the form of a guide tree as shown in Fig. 4.6b.

Problem 360 The hemoglobin proteins are homologs. In previous sections about *Adh* and its duplicate *Adh-dup* in *Drosophila*, we distinguished between two types of homologs: Orthologs, which have diverged through speciation, and paralogs, which have diverged through gene duplication. Classify our example genes into paralogs and homologs.

Problem 361 Save the four hemoglobin sequences in `hemoglobin.fasta` and align them using `clustalw`. Take a look at `hemoglobin.dnd`, which contains the guide tree in text format. To understand the relationship between a tree as text and a tree as graph, consider the guide tree in Fig. 4.6c. Its textual representation is



	A	B	C	D
A	0	6	6	2
B	6	0	4	6
C	6	4	0	6
D	2	6	6	0

Fig. 4.6 Alignment of multiple sequences as progressive pairwise alignment along a guide tree: Four sequences (**a**) are aligned pairwise and their distances stored in a matrix (**b**), which is summarized as tree (**c**); traversal of this tree from the leaves to the root guides the alignment (**d**–**f**)

```
((A:0.5,D:0.5):1,(B:1,C:1):0.5);
```

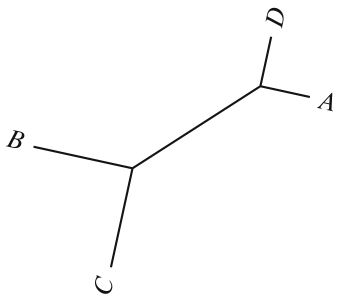
This notation means

Token	Explanation
`(`	cluster opened, draw a node that is either the root or an internal node
`A`	leaf with label *A*
`:0.5`	distance to parent node is 0.5
`,`	separate nodes
`)`	cluster closed
`;`	end of tree description

The program `clustalw` uses unrooted guide trees. Without a root our example tree would be denoted as

```
((A:0.5,D:0.5):1.5,(B:1,C:1));
```

and look like this



Draw by hand the guide tree produced by `clustalw`.

Problem 362 Take a look at the alignment in `hemoglobin.aln`. How many gaps does it contain, and during which phase of the algorithm was each of them introduced?

Problem 363 Let us examine the N-terminus of the alignment:

```
HBB_HORSE      -VQLS
HBB_HUMAN      MVHLT
HBA_HORSE      -MVLS
HBA_HUMAN      -MVLS
```

This contains one monomorphic site, `L`. However, the following arrangement of gaps gives two monomorphic sites, `L` and `V`:

```
HBB_HORSE      -VQLS
HBB_HUMAN      MVHLT
HBA_HORSE      MV-LS
HBA_HUMAN      MV-LS
```

Should not it be better than the first alignment? This depends on the treatment of gap-opening. By default, `clustalw` scores the opening of end gaps as zero. Consider the two sequences $S_1 = ATG$ and $S_2 = AG$. If gap opening is scored the same at every position, the optimal alignment should be

```
ATG
A-G
```

What does the `clustalw` alignment of these two sequences look like?

Problem 364 The gap-opening penalty of `clustalw` is set using

```
-GAPOPEN=x
```

Can you find a value for `x` that is compatible with the alternative N-terminal alignment:

```
HBB_HORSE     -VQLS...
HBB_HUMAN     MVHLT...
HBA_HORSE     MV-LS...
HBA_HUMAN     MV-LS...
```

Apply this to the hemoglobin sample.

Problem 365 To investigate the run time of `clustalw`, apply it to a single pair of random sequences of lengths 100, 200, 500, 1000, 2000, 5000, 10000, and 200000 bp. Plot the run times as a function of sequence length. How does the run time scale with sequence length?

Problem 366 Repeat the run time analysis, except this time vary the number of sequences aligned rather than their lengths. Generate 2, 5, 10, 20, 50, 100 random sequences of length 1 kb and measure the run times of `clustalw`. How does the run time scale with sample size?

Problem 367 Extract a random sample of 100 hemoglobin sequences from Uniprot/Swissprot (`uniprot_sprot.fasta`). For this, get the sequences with `getSeq` and convert them to tabular format by piping them through `fasta2tab.awk`. Shuffle the sequences using `shuffle.awk` to get an unbiased sample, then convert them back to FASTA format (`tr` and `fold`). How long does `clustalw` take to analyze this sample of 100 sequences? Repeat the time measurement with 200 random hemoglobin sequences.

Problem 368 How many hemoglobin sequences are contained in Uniprot? Estimate the time it would take to align all of them with `clustalw`.

Problem 369 Test your run time prediction.