GENOME-SCALE ALGORITHM DESIGN
by Veli Mäkinen, Djamal Belazzougui, Fabio Cunial and Alexandru I. Tomescu
Cambridge University Press, 2nd editions, 2023
www.genome-scale.info

Exercises for Chapter 6. Alignments

6.1 Show that the edit distance is a *metric*, namely that it satisfies:

(a) $D(A, B) \geq 0$,
(b) $D(A, B) = 0$, if and only if $A = B$,
(c) $D(A, B) = D(B, A)$,
(d) $D(A, B) \leq D(A, C) + D(C, B)$.

6.2 Modify Algorithm 6.1 on page 74 to use only one vector of length $m + 1$ in the computation.

6.3 The shortest-detour algorithm is able to compute the dynamic programming matrix only partially. However, just allocating space for the rectangular matrix of size $m \times n$ requires $O(mn)$ time. Explain how one can allocate just the cells that are to be computed. Can you do the evaluation in $O(d)$ space, where $d = D(A, B)$?

6.4 Modify Myers' bitparallel algorithm to solve the k-errors problem on short patterns.

6.5 Optimize Myers' bitparallel algorithm to use fewer basic operations.

6.6 Write a program to automatically verify each row in the truth tables in the lemmas proving the correctness of Myers' bitparallel algorithm.

6.7 Write a program to automatically discover the four "easy" operations (those of the four first lemmas) in Myers' bitparallel algorithm.

6.8 Describe how to traceback an optimal alignment with Myers' bitparallel algorithm.

6.9 Consider Exercise exe:hirschberg. Show that the same technique can be combined with Myers' bitparallel algorithm to obtain linear space traceback together with the bitparallel speed-up. *Hint.* Compute the algorithm also for the reverse of the strings. Then you can deduce $j_{\mathtt{mid}}$ from forward and reverse computations. See Section 7.4 for an analogous computation.

6.10 The four Russians technique we studied in Section 3.2 leads to an alternative way to speed up edit distance computation. The idea is to encode each column as in Myers' bitparallel algorithm with two bitvectors R^+_j and R^-_j and partition these bitvectors to blocks of length b. Each such pair of bitvector blocks representing an *output encoding* of a block $d_{i-b+1..i,j}$ in the original dynamic programming matrix is uniquely defined by the *input parameters* $A_{i-b+1..i}$, b_j, $d_{i-b,j-1}$, and $d_{i-b,j}$ and the bitvector blocks representing the block $d_{i-b+1..i,j-1}$. Develop the details and choose b so that one can afford to precompute a table storing for all possible input parameters the output encoding of a block. Show that this leads to an $O(mn/\log_\sigma n)$ time algorithm for computing the Levenshtein edit distance.

6.11 Prove Theorem 6.10.

6.12 Show that the set $M = M(A, B) = \{(i, j) \mid a_i = b_j\}$ which is needed for sparse dynamic programming LCS computation, sorted in reverse column-order, can be constructed in $O(\sigma + |A| + |B| + |M|)$ time on a constant alphabet and in $O((|A| + |B|) \log |A| + |M|)$ time on an ordered alphabet. Observe also that this construction can be run in parallel with Algorithm 6.3 to improve the space requirement of Theorem 6.12 to $O(m)$.

6.13 The sparse dynamic algorithm for distance $D_{\mathtt{id}}(A, B)$ can be simplified significantly if derived directly for computing $|\mathsf{LCS}(A, B)|$. Derive this algorithm. *Hint.* The search tree can be modified to answer range maximum queries instead of range minimum queries (see Section 6.4.5).

6.14 Give a pseudocode for local alignment using space $O(m)$.

6.15 Give a pseudocode for tracing an optimal path for the maximum-scoring local alignment, using space quadratic in the alignment length.

6.16 Develop an algorithm for tracing an optimal path for the maximum-scoring local alignment, using space linear in the alignment length. *Hint.* Let $[i'..i] \times [j'..j]$ define the rectangle containing a local alignment. Assume you know $j_{\mathtt{mid}}$ for row $(i - i')/2$ where the optimal alignment goes through. Then you can independently recursively consider rectangles defined by $[i'..(i - i')/2] \times [j'..j_{\mathtt{mid}}]$ and $[(i - i')/2] \times [j_{\mathtt{mid}}..j]$. To find $j_{\mathtt{mid}}$ you may, for example, store, along with the optimum value, the index j where that path crosses the middle row.

6.17 Prove the correctness of Algorithms 6.5 and 6.6.

6.18 The sequencing technology $\mathtt{SOLiD}^{\mathrm{TM}}$ from Applied Biosystems produces short reads of DNA in *color-space* with a two-base encoding defined by the following matrix (row = first base, column = second base):

	A	C	G	T
A	0	1	2	3
C	1	0	3	2
G	2	3	0	1
T	3	2	1	0

For example, `T012023211202102` equals `TTGAAGCTGTCCTGGA` (the first base is always given). Modify the overlap alignment algorithm to work properly in the case where one of the sequences is a `SOLiD` read and the other is a normal sequence.

6.19 Show that the relative entropy used in Equation (6.15) is non-negative.

6.20 Give a pseudocode for prokaryote gene alignment.

6.21 Give a pseudocode for eukaryote gene alignment with affine gaps.

6.22 Give a pseudocode for the $O(mn \cdot \mathtt{maxin})$ time algorithm for eukaryote gene alignment with limited introns.

6.23 Modify the recurrence for gene alignment with limited introns so that the alignment must start with a start codon, end with a stop codon, and contain `GT` and `AG` dinucleotides at intron boundaries. Can you still solve the problem in $O(mn \cdot \texttt{maxin})$ time? Since there are rare exceptions when dinucleotides at intron boundaries are something else, how can you make the requirement softer?

6.24 Give an alternative proof for the NP-hardness of the longest common subsequence problem on multiple sequences, by using a reduction from the vertex cover problem. *Hint.* From a graph $G = (V = \{1, \ldots, |V|\}, E)$ and integer k, construct $|E| + 1$ sequences having LCS of length $|V| - k$ if and only if there is vertex cover of size k in G. Recall that vertex cover V' is a subset of V such that all edges in E are incident to at least one vertex in V.

6.25 Develop a sparse dynamic programming solution for computing the longest common subsequence of multiple sequences. What is the expected size of the match set on random sequences from an alphabet of size σ? *Hint.* The search tree can be extended to higher-dimensional range queries using recursion (see the range trees from the computational geometry literature).

6.26 Reformulate the DAG-path alignment problem (Problem 6.19 on page 105) as a local alignment problem. Show that the Smith-Waterman approach extends to this, but is there a polynomial algorithm if one wishes to report alignments with negative score? Can the affine gap cost alignment be extended to the DAG-path alignment? Can one obtain a polynomial time algorithm for that case?

6.27 Show how to use DAG-path alignment to align a protein sequence to DNA. *Hint.* Represent the protein sequence as a *codon-DAG* replacing each amino acid by a sub-DAG representing its codons.

6.28 Consider a DAG with predicted exons as vertices and arcs formed by pairs of exons predicted to appear inside a transcript: this is the *splicing graph* considered in Chapter 15. Detail how DAG-path alignment on splicing graph and the codon-DAG of previous assignment solves eukaryote gene alignment problem. Show that the array $S[0..m]$ is not required to obtain the $O(|E|m)$ time approximate matching on graphs, so that the algorithm uses only $O(|V|)$ space.

6.29 Show how to traceback an optimal alignment using $O(\sqrt{m}|V|)$ space after conducting the $O(|E|m)$ time approximate matching on graphs.

6.30 Reformulate jumping alignment so that A is just aligned to the sequences forming the multiple alignment and not to the existing gaps inside the multiple alignment, as in Example 6.22. Show that the reformulated problem can be solved as fast as the original one.

Additional exercises not in the book

6.31 Compute the global alignment scores s_{ij} and especially $s_{mn} = S(A, B)$ for $A =$`ACCGATG` and $B =$`ACGGCTA` using indel penalty $-d$, where $d = 1$, and the substitution matrix:

$s(a,b)$	A	C	G	T
A	1	-1	-0.5	-1
C	-1	1	-1	-0.5
G	-0.5	-1	1	-1
T	-1	-0.5	-1	1

Trace an optimal alignment.

Rather than simulating on paper, you can also opt to implement. You may use the edit distance code at

`http://www.genome-scale.info/implementations/dp2tikz.py`

as a basis. The resulting TikZ-code can be included in a LaTeXdocument to visualize the result:

```
\documentclass{article}
\usepackage{tikz}

\begin{document}

\begin{tikzpicture}[scale=6.0]
\input{dp.tikz}
\end{tikzpicture}

\end{document}
```

6.32 Compute the local alignment scores $l_{i,j}$ for the same example as above. Trace an optimal local alignment.

6.33 In the affine gap model, the gaps are defined as runs of indels. Consider the alternative definition of gaps as runs of insertions or runs of deletions, briefly discussed in the main text. Derive the details of the basic recurrence to solve this variant.

6.34 Consider the setting above. Modify Gotoh's algorithm to handle the runs of insertions and runs of deletions separately. *Hint.* You will need three tables.

6.35 Consider the setting above. Modify the invariant-based algorithm to handle runs of insertions and runs of deletions separately.

6.36 Write a program (e.g. in Python) to read the k-th order distribution of a given DNA sequence (for given k), and to generate a new sequence of the same length simulating the output of this *k-th order Markov chain.*

6.37 `BLAST` is a heuristic aligner that makes local alignment feasible in large sequence databases. The following describes the main principles of `BLAST` with some simplifications. The database is indexed using a *k-mer index*, where each substring W of length k is associated with a list of pointers to the occurrence of W in the database. The lists of pointers of k-mers that are within Hamming distance 1 from some substring of a *query* sequence give the *candidate occurrences.* Dynamic programming is applied to extend candidate occurrences and to join nearby candidates, to form the final alignment results.

Show that `BLAST` is a *lossy filter*, meaning that it might miss some optimal local alignments.

6.38 Implement the k-mer index, e.g. by modifying your code for the k-th order Markov chain assignment above. The basic version is enough without considering how to compress the lists of occurrences (see Chapter 8. Classical indexes for more details about k-mer indexes).

6.39 Implement `BLAST`-like search on top of the k-mer index of the previous assignment to report candidate occurrences. You can ignore the dynamic programming part.

6.40 While there are many alternatives to `BLAST` that obtain better filtering accuracy, speed, and space, the popularity of `BLAST` is mostly explained by its handling of statistical significance of the found alignments. Rather than ranking alignments directly by maximum score, it takes into account the probability of finding equally good alignments by chance. Get familiar with this process by reading this tutorial `http://www.ncbi.nlm.nih.gov/BLAST/tutorial/`. Be ready to explain the main concepts.