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

Exercises for Chapter 13. Fragment assembly

13.1 Consider the following problem, called the Chinese postman problem. Given a directed graph $G = (V, E)$, a demand $d : E \to \mathbb{Z}_+$ for every arc, and a cost $c : E \to \mathbb{Q}_+$ for every arc, find a cycle in G (possibly with repeated vertices) visiting every arc (x, y) at least $d(x, y)$ times, and minimizing the sum of the costs of the arcs it traverses (note that if the cycle uses an arc (x, y) t times, then the cost associated with this arc is $tc(x, y)$). *Hint.* Reduce the problem to a minimum-cost flow problem.

13.2 An alternative formulation of correcting reads is to directly study the k-mer spectrum. Each k-mer having unexpectedly low coverage could be marked as *erroneous.* Consider a bipartite graph with erroneous k-mers on one side and the remaining k-mers on the other side. Add all edges between the two sides and define the edge cost between two k-mers as their edit distance. Find the minimum-cost maximum matching on this graph: recall Exercise 5.14 on page 66. How can you interprete this matching in order to correct the reads? Can you find a better formulation to exploit matching or network flows for error correction?

13.3 Fix some concrete implementation capable of implementing the replacement rules related to the read error correction after detection of bubbles in an assembly graph. What time and space complexities do you achieve?

13.4 Give a linear-time algorithm to find all bubbles in a directed graph.

13.5 Modify the linear-time bubble-finding algorithm from the previous assignment to work in small space using the succinct de Bruijn graph representations of Section 9.7. *Hint.* Use a bitvector to mark the already-visited vertices. Upon ending at a vertex with no unvisited out-neighbors, move the finger onto the bitvector for the next unset entry.

13.6 Consider replacing each bubble with one of its parallel unary paths. This can introduce new bubbles. One can iteratively continue the replacements. What is the worst-case running time of such iterative process?

13.7 Consider the above iterative scheme combined with the read error correction of Exercise 13.3. What is the worst-case running time of such an iterative read error correction scheme?

13.8 Given a directed graph $G = (V, E)$, let s and t be two vertices of G with the property that t is reachable from s, and denote by $U(s, t)$ the set of vertices reachable from s without passing through t. We say that $U(s, t)$ is a *superbubble* with *entrance* s and *exit* t if the following conditions hold:

- $U(s, t)$ equals the set of vertices from which t is reachable without passing through s;

- the subgraph of G induced by $U(s, t)$ is acyclic;
- no other vertex t' in $U(s, t)$ is such that $U(s, t')$ satisfies the above two conditions.

Write an $O(|V||E|)$-time algorithm for listing the entrance and exit vertices of all superbubbles in G.

13.9 Insight 13.2 introduced a special graph structure to cope with reverse complement reads in assembly graphs. Consider how maximal unary paths, tips, bubbles, and superbubbles should be defined on this special graph structure. Modify the algorithms for identifying these substructures accordingly.

13.10 Suppose you are given a set of d variable-length strings (reads) of total length N over the integer alphabet $[1..\sigma]$. Can you induce the sorted order of the strings in $O(N \log \sigma)$ time and bits of space (notice that the algorithm presented in Section 8.2.1 can sort in $O(N)$ time, but uses $O(N \log N)$ bots of space). *Hint.* Use some standard sorting algorithm, but without ever copying or moving the strings.

13.11 Recall the algorithm for computing maximal overlaps in Section 13.2.3. Fill in the details for constructing in linear time the tree of nested intervals from the suffix tree associated with intervals. *Hint.* First prove that the locus of $R^i[k..|R^i|]$ for any k is either a leaf or an internal node of depth $|R^i| - k + 1$.

13.12 Give the pseudocode for the algorithm sketched in Insight 13.3.

13.13 Analyze the running time and space of the algorithm in Insight 13.3. For this, you might find useful the concept of the *string graph*, which is like an overlap graph, but its arcs are labeled by strings as follows. An arc (i, j, ℓ) of an overlap graph becomes a bidirectional arc having two labels, $R^i[1..|R^i| - \ell]$ for the forward direction and $R^j[\ell + 1..|R^j|]$ for the backward direction. Observe that one can bound the running time of the algorithm in Insight 13.3 by the total length of the labels in the string graph before redundant arcs are removed.

13.14 Show that the following relaxation of Problem 13.1 remains NP-hard. Given a set $\mathcal{C}$ of contigs, the alignments of a set $\mathcal{R}$ of paired-end reads in these contigs, and integer t, find a subset $\mathcal{R}' \subseteq \mathcal{R}$ of paired-end reads such that $\mathcal{R}'$ has the property that the contigs in $\mathcal{C}$ can be partitioned into the minimum number of scaffolds such that

- every read in $\mathcal{R}'$ is t-consistent with some scaffold, and
- each scaffold is connected with respect to $\mathcal{R}'$.

13.15 Formulate different problem statements, similar to Problem 13.1, for taking into account the fact that a contig may belong to multiple scaffolds. What can you say about their computational complexity?

13.16 Problem 13.1 could be modified to ask for the minimum k read pairs to remove such that the remaining read pairs form a consistent scaffolding. Study the computational complexity of this problem variant.

13.17 We gave a pseudo-polynomial algorithm for gap filling (Problem 13.2), in the sense that if the gap length parameter d is bounded (by say a polynomial function in n and m), then the algorithm runs in polynomial time (in n and m). We left open the hardness of the problem when d is unbounded. Consider a generalization of the problem when the input graph is any weighted graph (with any non-negative arc costs), not just a prefix graph constructed from a collection of strings. Show that this generalized problem is NP-complete, by giving a reduction from the *subset sum* problem (recall its definition from the proof of Theorem 5.3) to this generalization (use some arcs with 0 cost). Why is it hard to extend this reduction to the specific case of prefix graphs?

13.18 Consider the following generalization of Problem 13.2 in which we allow approximate overlaps. Given a prefix graph $G = (V, E)$, with a cost function $c : E \to \mathbb{Z}_+$ and $h : E \to \mathbb{Z}_+$, two of its vertices s and t, and an interval of possible path costs $[d'..d]$, decide whether there is a path P made up of $s = v_1, v_2, \ldots, v_k = t$ such that

$$\mathsf{cost}(P) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}) \in [d'..d],$$

and among all such paths return one P minimizing

$$\sum_{i=1}^{k-1} h(v_i, v_{i+1}).$$

Show how to solve this problem in time $O(dm)$. What is the space complexity of your algorithm?

13.19 Modify the solution to Problem 13.2 so that it runs in time $O(m_r(d + \log r))$, where r is the number of vertices reachable from s by a path of cost at most d, and m_r is the number of arcs between these vertices.

13.20 Modify the solution to Problem 13.2 so that we obtain also the number of paths from s to t whose cost belongs to the interval $[d'..d]$.

13.21 Interpret the dynamic programming algorithm given for Problem 13.2 as an algorithm working on a DAG, constructed in a similar manner to that in Section 4.2.2 for the Bellman–Ford method.

13.22 Consider a de Bruijn graph G built on a set $\mathcal{R}$ of reads sequenced from an unknown genome, and let P be a unary path in G. Under which assumptions on $\mathcal{R}$ the unary path P spells a string occurring in the original genome?

13.23 Let $G = (V, E)$ be a directed graph. Give an $O(|V| + |E|)$-time algorithm for outputting all maximal unary paths of G.

13.24 Let $G = (V, E)$ be a directed graph, and let G' be the graph obtained from G by compacting its arcs through the following process. First, label every vertex v of G with the list $l(v) = (v)$, that is, $l(v)$ is made up only of v. Then, as long as there is an arc (u, v) such that $N^+(u) = \{v\}$ and $N^-(v) = \{u\}$, then remove the vertex v, append v at the end of $l(u)$, and add arcs from u to every out-neighbor of v. Explain how, given only G', it is possible to report all maximal unary paths of G in time $O(|V| + |E|)$. What is the time complexity of computing G'?

13.25 Let $G = (V, E)$ be a directed graph. A more formal way of defining a contig is to say that a path P is a *contig* if P appears as a subpath of any path in G which covers all vertices of G.

- Let P be a unary path of length at least 2. Show that P is a contig.
- Let $P = (s, v_1, \ldots, v_k, \ldots, v_n, t)$ be a path such that v_k is an unary vertex, for all $1 \leq i < k$, we have $|N^-(v_i)| = 1$, and for all $k < j \leq n$, we have $|N^+(v_j)| = 1$. Show that P is a contig.

Additional exercises not in the book

13.26 Suppose that for an assembly graph you know, for every vertex v, the number $|N^-(v)|$ of its in-neighbors, and the number $|N^+(v)|$ of its out-neighbors. Come up with a simple formula for counting the number of maximal unary paths of the graph without actually traversing the arcs of the graph and reporting its maximal unary paths.