15 Transcriptomics

Recall from Chapter 1 that, through transcription and alternative splicing, each gene produces different RNA transcripts. Depending on various factors, such as the tissue the cell is in, owing to disease, or in response to some stimuli, the RNA transcripts of a gene and the number of copies produced (their *expression level*) can be different.

In this chapter we assume that we have a collection of reads from all the different (copies of the) transcripts of a gene. We also assume that these reads have been aligned to the reference genome, using for example techniques from Section 10.6; in addition, Section 15.4 shows how to exploit the output of genome analysis techniques from Chapter 11 to obtain an aligner for long reads of RNA transcripts. Our final goal is to assemble the reads into the different RNA transcripts, and to estimate the expression level of each transcript. The main difficulty of this problem, which we call *multi-assembly*, arises from the fact that the transcripts share identical substrings.

We illustrate different scenarios, and corresponding multi-assembly formulations, stated and solved for each individual gene. In Section 15.1 we illustrate the simplest one, in which the gene's transcripts are known in advance, and we need only find their expression levels from the read alignments. In Section 15.2 we illustrate the problem of assembling the RNA reads into the different unknown transcripts, without estimating their levels of expression. In Section 15.3 we present a problem formulation for simultaneous assembly and expression estimation.

As just mentioned, in this chapter we assume that we have a reference genome, and thus that we are in a so-called *genome-guided* setting. De novo multi-assembly is in general a rather hard task. Thus, we prefer here to stick to genome-guided multi-assembly, which admits clean problem formulations and already illustrates the main algorithmic concepts. Nevertheless, in Insight 15.1 we briefly discuss how the least-squares method from Section 15.1 could be applied in a de novo setting.

15.1 Estimating the expression of annotated transcripts

Recall from Chapter 1 that each gene can be seen as partitioned into exons (substrings that appear in at least one transcript) and introns (substrings that are removed through transcription and do not appear in any transcript). In this section we assume that we have so-called *gene annotation*, meaning that we have a list T_i of exon starting and ending positions, for each of the k transcripts of a gene. Given a collection of reads sequenced

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

from some of these annotated transcripts, we want to discover the *expression level* of each transcript, that is, how many copies of it appear in the sample.

First, we align the reads to the annotated transcripts, either by concatenating the annotated exons and aligning the reads to the resulting genomic sequences (using techniques from Chapter 10), or by split alignment of the reads directly to the genome (using techniques from Section 10.6).

Second, we construct a directed graph G whose vertices are the annotated exons, and whose arcs correspond to pairs of exons consecutive in some annotated transcript. The resulting graph G is called a *splicing graph*, because it models the alternative splicing of the transcripts. Moreover, G is a DAG, since the direction of its arcs is according to the position in the reference genome of its two endpoint exons. The annotated transcripts $T_1, \ldots, T_k$ correspond to paths $P_1, \ldots, P_k$ in G. For each annotated exon, namely for each vertex of G, we can compute an average read coverage. Similarly, we can compute the number of reads overlapping each splice-site alternative, that is, each position where two exons are concatenated, and thus each arc of G.

A reasonable assumption on how transcript sequencing behaves is that the coverage does not depend on the position inside the transcript. Observe, however, that, if the reads are of length m, the transcript coverage in the first and last $m-1$ positions of each transcript presents an upward and downward slope, respectively. In this chapter we assume that such artifacts have been appropriately addressed, so that each annotated transcript T_i has an unknown average coverage e_i, its expression level. In the following two sections, where we assume that gene annotation is not available, this upward/downward slope is actually helpful in identifying potential start and end exons.

We can formulate this problem as follows. Given paths $P_1, \ldots, P_k$ in G (the annotated transcripts), find the expression level e_i of each path P_i, minimizing the sum of squared differences between the observed coverage of a vertex or an arc and the sum of the expression levels of the paths using that vertex or arc.

We make a simplifying reduction from this problem to one in which the input graph has coverages associated only with its arcs, as was done also in Section 5.4.2. This can be achieved by subdividing each vertex v into two vertices v_{in} and v_{out}, and adding the arc (v_{in}, v_{out}) having the same coverage as v. We also make all former in-neighbors of v in-neighbors of v_{in}, and all former out-neighbors of v out-neighbors of v_{out}. We use this simplified problem input henceforth, and later in Section 15.3.

Problem 15.1 Annotated transcript expression estimation

Given a directed graph $G = (V, E)$, a weight function $w : E \to \mathbb{Q}_+$, and a collection of k directed paths $P_1, \ldots, P_k$ in G, find expression levels $e_1, \ldots, e_k$ for each path, minimizing the following sum of errors:

$$\sum_{(x,y)\in E} \left(w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right)^2 . \tag{15.1}$$

---

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Problem 15.1 is a *least-squares problem*, and can be solved analytically. The idea is to reduce the problem to solving a system of *linear* equations, which reduction is possible thanks to the particular definition of the objective function, as a sum of *squared* errors. See Insight 15.3 on page 342 for a discussion on different error functions.

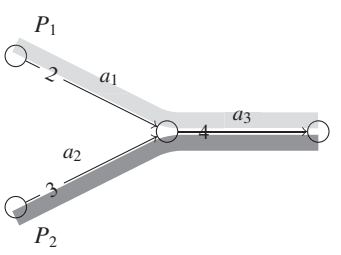
We introduce some further notation here. We assume that the set E of arcs of G is $\{a_1, \ldots, a_m\}$, and that each a_i has weight w_i. Moreover, for every $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, k\}$, let $\alpha_{i,j}$ be indicator variables defined as

$$\alpha_{i,j} = \begin{cases} 1, & \text{if arc } a_i \text{ belongs to path } P_j, \\ 0, & \text{otherwise.} \end{cases}$$

Using the indicator variables $\alpha_{i,j}$, we can rewrite (15.1) as

$$\begin{aligned} f(e_1, \ldots, e_k) = &\left(w_1 - \alpha_{1,1}e_1 - \alpha_{1,2}e_2 - \cdots - \alpha_{1,k}e_k\right)^2 \\ &+ \left(w_2 - \alpha_{2,1}e_1 - \alpha_{2,2}e_2 - \cdots - \alpha_{2,k}e_k\right)^2 \\ &\cdots \\ &+ \left(w_m - \alpha_{m,1}e_1 - \alpha_{m,2}e_2 - \cdots - \alpha_{m,k}e_k\right)^2. \end{aligned} \tag{15.2}$$

The function f is a function in k variables $e_1, \ldots, e_k$; it is quadratic in each variable. See Figure 15.1 for an example in two variables. It attains its global minimum in any point satisfying the system of equations obtained by equating to 0 each partial derivative of $f(e_1, \ldots, e_k)$ with respect to each variable. We thus obtain the following system of equations:



$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 6 \\ 7 \end{pmatrix}$$

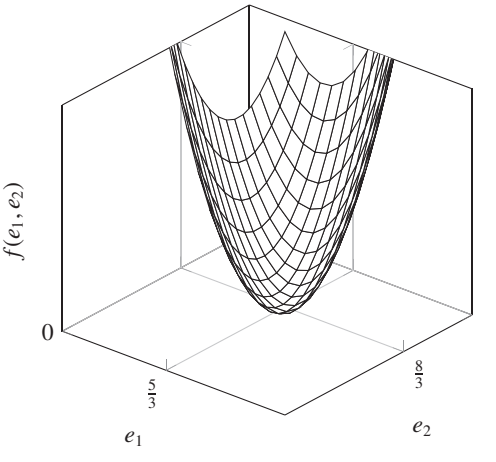


Figure 15.1 On the left, a directed graph G with arcs a_1, a_2, and a_3, having weights 2, 3, and 4, respectively. P_1 and P_2 are two directed paths in G. The indicator matrix A is shown, as well as the corresponding system of equations (15.4). On the right is shown the function $f(e_1, e_2)$, having a unique minimum at $\left(\frac{5}{3}, \frac{8}{3}\right)$, the unique solution to the system (15.4).

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

$$\begin{aligned}
e_1 \sum_{j=1}^{m} \alpha_{j,1}\alpha_{j,1} + \cdots + e_k \sum_{j=1}^{m} \alpha_{j,1}\alpha_{j,k} &= w_1\alpha_{1,1} + \cdots + w_m\alpha_{m,1}, \\
e_1 \sum_{j=1}^{m} \alpha_{j,2}\alpha_{j,1} + \cdots + e_k \sum_{j=1}^{m} \alpha_{j,2}\alpha_{j,k} &= w_1\alpha_{1,2} + \cdots + w_m\alpha_{m,2}, \\
&\cdots \\
e_1 \sum_{j=1}^{m} \alpha_{j,k}\alpha_{j,1} + \cdots + e_k \sum_{j=1}^{m} \alpha_{j,k}\alpha_{j,k} &= w_1\alpha_{1,k} + \cdots + w_m\alpha_{m,k}.
\end{aligned} \tag{15.3}$$

(Exercise 15.1 asks the reader to fill in the derivations needed for obtaining this system.)

We rewrite (15.3) more compactly using matrix notation. Let $A_{1..m,1..k}$ be a matrix such that $A[i,j] = \alpha_{i,j}$, for all $i \in \{1,\ldots,m\}$, $j \in \{1,\ldots,k\}$. Let A^{T} be its transpose, namely A^{T} is the $k \times m$ matrix such that $A[i,j] = \alpha_{j,i}$, for all $i \in \{1,\ldots,k\}, j \in \{1,\ldots,m\}$. A solution $(e_1,\ldots,e_k)$ for Problem 15.1 is any solution to the system of equations

$$A^{\mathrm{T}}A \begin{pmatrix} e_1 \\ \vdots \\ e_k \end{pmatrix} = A^{\mathrm{T}} \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix}, \tag{15.4}$$

(recall that A and $w_1,\ldots,w_m$ are known). Observe that system (15.4), and thus Problem 15.1, always admits a solution since the matrix $A^{\mathrm{T}}A$ is a square matrix of size k. Moreover, the solution is unique if and only if $\det(A^{\mathrm{T}}A) \neq 0$.

The matrix $A^{\mathrm{T}}A$ can be computed in time $O(mk^2)$, since each of the k^2 elements of $A^{\mathrm{T}}A$ can be computed in time $O(m)$. The system (15.4) can be solved for example by Gaussian elimination, which requires in total $O(k^3)$ arithmetic operations.

Insight 15.1 De novo annotated transcript expression estimation

Observe that the solution for Problem 15.1 does not require the input directed graph to be acyclic. Therefore, we could try applying it to a de novo setting, where G is the de Bruijn graph of a set of reads. For each arc, representing a k-mer, we can store the number of reads containing that k-mer. The annotated transcripts that we now take as input are strings, which need to be aligned to G, combining methods from Section 9.7 and Chapter 10; namely, one can slide a k-mer window through a transcript sequence, navigate the de Bruijn graph, and branch in the search to allow a limited amount of alignment errors.

Therefore, from the alignment of each transcript T_i we can again obtain a path P_i picking up the vertices belonging to the alignment (P_i can possibly have repeated vertices) in G. Thus, we are back in the same setting as in Problem 15.1, with the exception that the input graph might not be acyclic.

Observe that the complexity $O(mk^2)$ (where m now denotes the number of distinct k-mers) of computing the matrix $A^{\mathrm{T}}A$ is still linear in m, and finding a solution to the corresponding system of equations has a complexity independent of m.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

15.2 Transcript assembly

We now study the problem of finding the most likely transcripts of a gene, given only the RNA split-read alignments. Thus, in contrast to the previous section, we assume here that we do not have any gene annotation. We also assume that we just need to find the transcripts, without estimating their expression levels.

Even without existing gene annotation, the exons of a gene can be discovered from split-read alignments, by observing that

- coverage information helps in discovering the start position of the first exon in a transcript and the end position of the last exon of a transcript, which can be marked as virtual splice-sites;
- the exons and introns of a gene are then contiguous stretches of the reference genome between two consecutive splice-sites; the exons are those stretches with coverage above a given confidence threshold.

These two steps can be formalized and implemented in the same manner as in Insight 14.1 for peak detection using HMMs. Exercise 15.2 asks the reader to formalize this exon detection phase using HMMs. The exons give the set of vertices of the splicing graph. The arcs of the splicing graph are similarly derived from reads overlapping two exons, indicating that they are consecutive in some transcript. Henceforth we assume that we have already constructed a splicing directed acyclic graph G.

The main difference in this section with respect to the rest of this chapter is that we do not try to explain the observed coverages of the splicing graph in terms of a least-squares model, but instead associate a cost with each arc between two vertices u and v, as the belief that u and v originate from *different* transcripts. Accordingly, we want to find a collection of paths that will cover all of the exons, at minimum total cost.

We should underline that the algorithms which we will present in this section depend only on the acyclicity of the splicing graph. As a matter of fact, the input could also be an overlap graph (whose vertices stand for reads and whose arcs stand for overlaps between reads – recall Section 13.2.3), since in this case the graph is also acyclic, because the reads have been aligned to the reference.

15.2.1 Short reads

In this section we assume that we have only a collection of "short" reads, in the sense that they overlap at most two consecutive exons. If a read happens to overlap more exons, then we can consider each overlap over two consecutive exons as a separate short read.

It is common practice to impose a *parsimony* principle in this multi-assembly problem, and to require the simplest interpretation that can explain the data. Thus, among all possible collections of transcripts covering all the exons of a gene, one with the minimum number of transcripts is preferred. Among all such minimum solutions, the one with the minimum total cost is preferred.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

This problem is that of the minimum-cost minimum path cover, studied in Section 5.4.2 as Problem 5.11. However, we make one amendment to it here. Observe that, from the splicing graph construction phase, the read coverage indicates exons that are likely candidates to be start vertices of the solution paths. Likewise, we also know which exons are candidates for the end vertices of the solution paths. Therefore, we have the following problem.

Problem 15.2 Transcript assembly with short reads

Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, and cost function $c : E \to \mathbb{Q}_+$, find the minimum number k of paths $P_1, \ldots, P_k$ such that

- every $v \in V$ belongs to some P_i (that is, $P_1, \ldots, P_k$ form a path cover of G),
- every P_i starts in some vertex of S, and ends in some vertex of T, and

among all path covers with k paths satisfying the above condition, they minimize

$$\sum_{i=1}^{k} \sum_{(x,y)\in P_i} c(x, y).$$

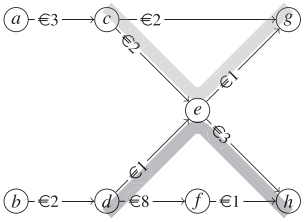
Henceforth we assume that all sources of G belong to S and all sinks of G belong to T, since otherwise Problem 15.2 admits no solution. Problem 15.2 can be solved as was done in Section 5.4.2: we construct the same flow network $N = (G^*, \ell, u, c, q)$, with the only modification that the global source s^* is connected to every vertex v_{in}, with $v \in S$ (instead of every source v of G), and every vertex v_{out}, with $v \in T$ (instead of every sink v of G), is connected to the global sink t^*. Computing the minimum-cost flow on N and splitting it into paths gives the solution paths. Thus, we analogously obtain the following theorem.

THEOREM 15.1 *The problem of transcript assembly with short reads on a DAG G with n vertices, m arcs, and positive costs is solvable in time $O(nm + n^2 \log n)$.*

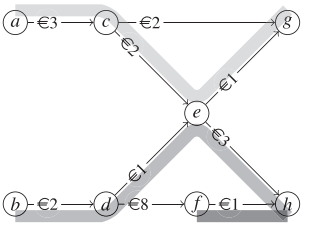
15.2.2 Long reads

A splicing graph, through its arcs, models constraints only on pairs of exons that must be consecutive in some transcript. However, if the reads are long enough (or if some exons are short enough) a read can overlap multiple exons. We will consider in Section 15.4 how to find such multiple-exon read overlaps. Such overlaps give rise to further constraints on the admissible solutions to a transcript assembly problem. One way to model this problem of transcript assembly with long reads is to receive in the input also a collection of paths in the splicing graph (the long reads) and to require that every such path be completely contained in some solution path. Thus, we extend Problem 15.2 to Problem 15.3 below. See Figure 15.2 for an example.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press



(a) An input DAG G, with $S = \{a, b, c, f\}$ and $T = \{g, h\}$. The two constraint paths P_1^{in} and P_2^{in} are shown with thick gray lines.



(b) The solution for G consists of $k = 3$ paths, shown with thick gray lines.

Figure 15.2 An example for the problem of transcript assembly with long reads.

Problem 15.3 Transcript assembly with long reads

Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, a cost function $c : E \to \mathbb{Q}_+$, and a collection $\mathcal{P}^{in} = \{P_1^{in}, \ldots, P_t^{in}\}$ of directed paths in G, find the minimum number k of paths $P_1^{sol}, \ldots, P_k^{sol}$ forming a path cover of G, such that

- every P_i^{sol} starts in some vertex of S, and ends in some vertex of T,
- every path $P^{in} \in \mathcal{P}^{in}$ is entirely contained in some P_i^{sol}, and

among all path covers with k paths satisfying the above conditions, they minimize

$$\sum_{i=1}^{k} \sum_{(x,y) \in P_i^{sol}} c(x, y).$$

We solve Problem 15.3 also by reducing it to a network flow problem. The idea is to represent each path constraint P^{in} starting in a vertex u and ending in a vertex v by an arc (u, v) with demand 1 in the corresponding flow network N. The other arcs of N corresponding to vertices of P^{in} receive demand 0, since, if we cover P^{in}, we have already covered all of its vertices. However, the problematic cases (see Figure 15.3) are when a path constraint is completely contained in another one, or when the suffix of a path constraint equals the prefix of another path constraint. This is because adding such arcs between their endpoints increases the number k of paths in a solution for Problem 15.3.

We solve this by preprocessing the input constraints with the following two steps.

Step 1. While there are paths $P_i^{in}, P_j^{in} \in \mathcal{P}^{in}$ such that P_i^{in} is contained in P_j^{in}, remove P_i^{in} from $\mathcal{P}^{in}$.

If a path P_i^{in} is completely contained in another path P_j^{in}, then if we cover P_j^{in} with a solution path, this solution already covers P_i^{in}. This is key also for the correctness of the next step.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

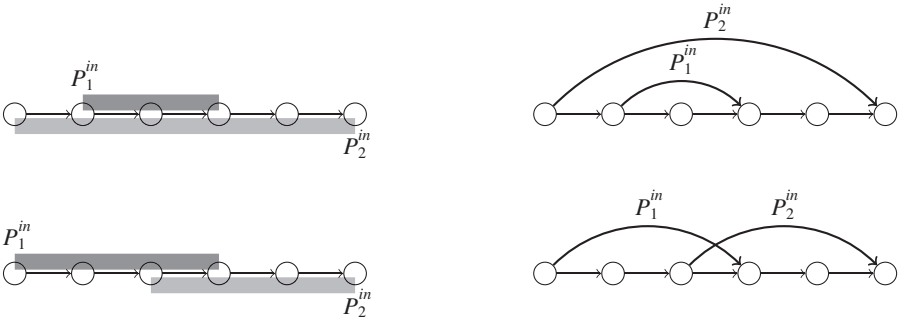


Figure 15.3 Problematic cases in reducing the problem of transcript assembly with long reads to a network flow problem. They can be solved after preprocessing the collection $\mathcal{P}^{in}$ of path constraints by Steps 1 and 2. Top, a path constraint P_1^{in} completely included in P_2^{in}; bottom, path constraints P_1^{in} and P_2^{in} have a suffix–prefix overlap.

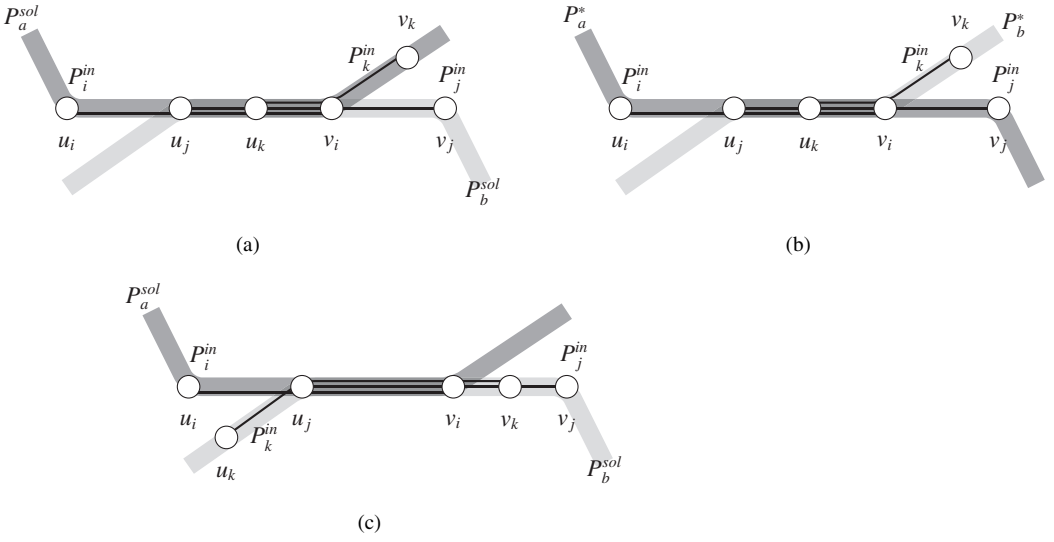


Figure 15.4 A visual proof of Lemma 15.2. Subpath constraints are drawn with thin black lines, solution paths are drawn with thick gray lines.

Step 2. While there are paths $P_i^{in}, P_j^{in} \in \mathcal{P}^{in}$ such that a suffix of P_i^{in} is a prefix of P_j^{in} do the following:

- let $P_i^{in}, P_j^{in} \in \mathcal{P}^{in}$ be as above, with the *longest* suffix–prefix overlap;
- let P_{new}^{in} := the path $P_i^{in} \cup P_j^{in}$ which starts as P_i^{in} and ends as P_j^{in};
- $\mathcal{P}^{in} := \left(\mathcal{P}^{in} \setminus \{P_i^{in}, P_j^{in}\}\right) \cup \{P_{new}^{in}\}$.

In this step, we merge paths having a suffix–prefix overlap. We do this iteratively, at each step merging that pair of paths for which the suffix–prefix overlap is the longest possible.

The correctness of these two steps is guaranteed by the following lemma.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

LEMMA 15.2 *If the problem of transcript assembly with long reads for an instance* $(G, \mathcal{P}^{in})$ *admits a solution with k paths, then also the problem instance transformed by applying Steps 1 and 2 admits a solution with k paths, and this solution has the same cost and also satisfies the original constraints* $\mathcal{P}^{in}$.

Proof The correctness of Step 1 is clear. Assume that $\mathcal{P}^{in}$ has been transformed by Steps 1 and 2, and let $P_i^{in}, P_j^{in} \in \mathcal{P}^{in}$ be such that their suffix–prefix overlap is the *longest* possible. Suppose that the original problem admits a solution $\mathcal{P}^{sol} = \{P_1^{sol}, \ldots, P_k^{sol}\}$ such that P_i^{in} and P_j^{in} are covered by different solution paths, say P_a^{sol} and P_b^{sol}, respectively. We show that the transformed problem admits a solution $\mathcal{P}^* = \left(\{P_1^{sol}, \ldots, P_k^{sol}\} \setminus \{P_a^{sol}, P_b^{sol}\}\right) \cup \{P_a^*, P_b^*\}$ having the same cardinality as $\mathcal{P}$, in which P_i^{in} and P_j^{in} are covered by the same path P_a^*, and $\mathcal{P}^*$ also satisfies all of the original constraints $\mathcal{P}^{in}$.

Suppose that P_i^{in} starts with vertex u_i and ends with vertex v_i, and that P_j^{in} starts with vertex u_j and ends with vertex v_j.

- Let P_a^* be the path obtained as the concatenation of the path P_a^{sol} taken from its starting vertex until v_i with the path P_b^{sol} taken from v_i until its end vertex (so that P_a^* covers both P_i^{in} and P_j^{in}).
- Let P_b^* be the path obtained as the concatenation of the path P_b^{sol} taken from its starting vertex until v_i with the path P_a^{sol} taken from v_i until its end vertex.

See Figures 15.4(a) and (b). We have to show that the collection of paths $\mathcal{P}^* = \left(\{P_1^{sol}, \ldots, P_k^{sol}\} \setminus \{P_a^{sol}, P_b^{sol}\}\right) \cup \{P_a^*, P_b^*\}$ covers all the vertices of G, has the some cost as $\mathcal{P}^{sol}$ and satisfies the original constraints $\mathcal{P}^{in}$. Since P_a^* and P_b^* use exactly the same arcs as P_a^{sol} and P_b^{sol}, $\mathcal{P}^*$ is a path cover of G, having the some cost as $\mathcal{P}^{sol}$.

The only two problematic cases are when there is a subpath constraint P_k^{in} that has v_i as internal vertex and is satisfied only by P_a^{sol}, or it is satisfied only by P_b^{sol}. Denote, analogously, by u_k and v_k the endpoints of P_k^{in}. From the fact that the input was transformed by Step 1, P_i^{in} and P_j^{in} are not completely included in P_k^{in}.

Case 1. P_k^{in} is satisfied only by P_a^{sol} (Figures 15.4(a) and (b)). Since P_i^{in} is not completely included in P_k^{in}, u_k is an internal vertex of P_i^{in}; thus, a suffix of P_i^{in} is a prefix also of P_k^{in}. From the fact that the overlap between P_i^{in} and P_j^{in} is the longest possible, we have that vertices u_j, u_k, and v_i appear in this order in P_i^{in}. Thus, P_k^{in} is also satisfied by P_b^*, since u_k appears after u_j on P_i.

Case 2. P_k^{in} is satisfied only by P_b^{sol}, and it is not satisfied by P_a^* (Figure 15.4(c)). This means that P_k^{in} starts on P_b^{sol} before u_j and, since it contains v_i, it ends on P_b^{sol} after v_i. From the fact that P_j^{in} is not completely included in P_k^{in}, v_k is an internal vertex of P_j^{in}, and thus a suffix of P_k^{in} equals a prefix of P_j^{in}. This common part is now longer than the common suffix/prefix between P_i^{in} and P_j^{in}, which contradicts the maximality of the suffix/prefix between P_i^{in} and P_j^{in}. This proves the lemma. □

We are now ready to reduce this problem to a minimum-cost flow problem. As was done in Section 5.4.2 for the minimum path cover problem, and in the previous section, we construct a flow network $N = (G^*, \ell, u, c, q)$ (see Figure 15.5).

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

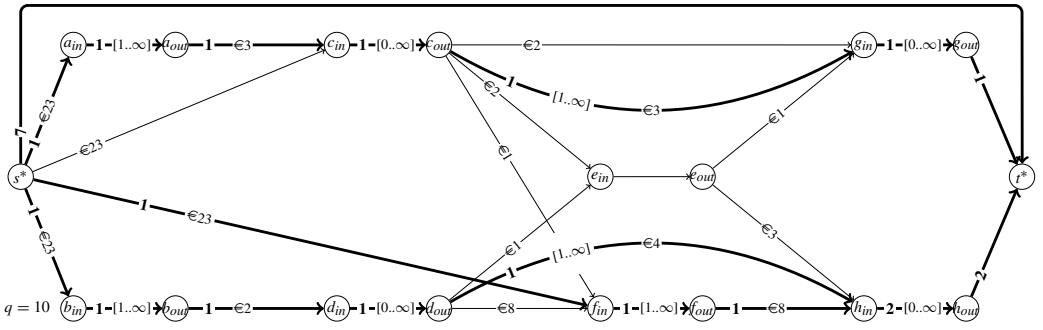


Figure 15.5 The flow network N constructed from the graph in Figure 15.2(a). Arcs without cost labels have null cost; arcs without demand labels have null demand. Arcs with non-null value in a minimum-cost flow over N are highlighted.

- For every vertex x of G, we introduce two vertices x_{in} and x_{out} in G^*, and add the arc (x_{in}, x_{out}) to G^*, with null cost;
- for every arc (y, x) incoming into x, we add the arc (y_{out}, x_{in}) to G^*, with cost $c(y, x)$;
- for every arc (x, y) outgoing from x, we add the arc (x_{out}, y_{in}) to G^* with cost $c(x, y)$.

We add a global source s^* with arcs to every vertex x_{in}, if $x \in S$. Likewise, we add a global sink t^* and arcs (x_{out}, t^*), for every sink $x \in T$.

For every path $P^{in} \in \mathcal{P}^{in}$ ($\mathcal{P}^{in}$ is transformed by Steps 1 and 2), starting in u and ending in v, we add the arc (u_{out}, v_{in}). We set its cost $c(u_{out}, v_{in})$ as the sum of the costs of all arcs of P^{in}, and we set its demand to 1. Moreover, for every arc (v_{in}, v_{out}) of G^* such that v belongs to no path constraint in $\mathcal{P}^{in}$, we set its demand to 1. All other arcs have demand 0.

We set the value of the flow to $q = |V| + |P^{in}|$, since at most $|V| + |P^{in}|$ paths are needed in any solution for Problem 15.3. To account for the supplementary flow, we also add the arc (s^*, t^*) with cost 0 and demand 0.

Finally, we set the cost of each arc (s^*, x_{in}) to 1 plus the sum of all other costs of G^*, since we want a solution with the minimum number of paths. A minimum-cost flow f induces a path cover in G in the following way. Since G^* is acyclic, we can consider a decomposition of f minus the arc (s^*, t^*) into the k weighted paths $P_1, \ldots, P_k$ from s^* to t^* (all having weight 1, from the minimality of f). We remove the vertices s^* and t^* from each P_i, and contract the arcs of the form (x_{in}, x_{out}) back into x. We obtain a collection $Q_1, \ldots, Q_k$ of paths in G^*. Moreover, whenever some Q_i uses an arc (u_{out}, v_{in}) corresponding to a constraint path P^{in}, we replace it in Q_i with P^{in}. This gives a collection $R_1, \ldots, R_k$ of paths in G, which is the solution to Problem 15.3.

Therefore, we have the following result.

THEOREM 15.3 *The problem of transcript assembly with long reads on a graph with n vertices, m arcs, and c subpath constraints, with K being the sum of subpath constraint*

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

lengths, can be solved by reducing it to a minimum-cost circulation problem on a network with $O(n)$ vertices and $O(m + c)$ arcs, with demands only. This network can be computed in time $O(K + c^2 + m)$, and the complexity of the problem of transcript assembly with long reads becomes $O(K + n^2 \log n + n(m + c))$.

Proof Network N has $O(n)$ vertices and $O(m + c)$ arcs. The minimum-cost flow problem on N can be reduced to a minimum-cost circulation problem on a network obtained from N by adding the arc (t^*, s^*), with cost 0 and demand 0. This network has demands only, and they all equal 1, thus the minimum-cost circulation can be computed in time $O(n^2 \log n + n(m + c))$.

We now analyze the complexity of the preprocessing phase. If implemented naively, it takes $O(c^2 n^2)$ time. However, this can be lowered to $O(K + \texttt{output})$, where $\texttt{output} \leq c^2$ is the number of pairs having an overlap.

Step 1 can be solved by first building a (generalized) suffix tree on the concatenation of path constraints with a distinct symbol $\#_i$ added after each constraint sequence P_i^{in}. This can be done in $O(K)$ time even on our alphabet of size $O(n)$ by Theorem 8.15 on page 143. Then one can do as follows during depth-first traversal of the tree: if a leaf corresponding to the suffix starting at the beginning of subpath constraint P_i^{in} has an incoming arc labeled only by $\#_i$ and its parent still has other children, then the constraint is a substring of another constraint and must be removed (together with the leaf).

For Step 2, we compute all pairs of longest suffix–prefix overlaps between the subpath constraints using the $O(K + \texttt{output})$ time algorithm in Theorem 8.22. The output can be put into the form of a list L containing elements of the form $(i, j, \texttt{len})$ in decreasing order of the overlap length, `len`, between constraints P_i^{in} and P_j^{in}. Notice that each constraint must take part in only one merge as the first element, and in one merge as the second element, since merging cannot produce completely new overlaps; otherwise some constraint would be included in another constraint, which is not possible after Step 1. We can thus maintain two bitvectors $F[1..c]$ and $S[1..c]$ initialized to zeros, such that, once P_i^{in} and P_j^{in} have been merged, we set $F[i] = 1$ and $S[j] = 1$. That is, we scan list L from beginning to end, apply the merge if the correspondong bits are not set in F and S, and set the bits after each merge.

Merging itself requires a linked list structure: all the constraints are represented as doubly-linked lists with vertex numbers as elements. Merging can be done by linking the doubly-linked lists together, removing the extra overlapping part from the latter list, and redirecting its start pointer to point inside the newly formed merged list. When one has finished with merging, the new constraints are exactly those old constraints whose start pointers still point to the beginning of a vertex list. The complexity of merging is thus $O(K)$. □

15.2.3 Paired-end reads

As paired-end reads are sequenced from ends of the same fragment of RNA, they also impose constraints on sequences of vertices that must appear together in the same assembled transcript. Thus, we can further generalize Problems 15.2 and 15.3 into

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Problem 15.4 below, in which we are given a collection of pairs of paths, and we require that both paths in a pair are fully covered by the same solution path.

Problem 15.4 Transcript assembly with paired-end reads

Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, a cost function $c : E \to \mathbb{Q}_+$, and a collection $\mathcal{P}^{in} = \{(P^{in}_{1,1}, P^{in}_{1,2}), \ldots, (P^{in}_{t,1}, P^{in}_{t,2})\}$ of pairs of directed paths in G, find the minimum number k of paths $P^{sol}_1, \ldots, P^{sol}_k$ forming a path cover of G, such that

- every P^{sol}_i starts in some vertex of S, and ends in some vertex of T,
- for every pair $(P^{in}_{j,1}, P^{in}_{j,2}) \in \mathcal{P}^{in}$, there exists P^{sol}_i such that both $P^{in}_{j,1}$ and $P^{in}_{j,2}$ are entirely contained in P^{sol}_i, and

among all path covers with k paths satisfying the above conditions, they minimize

$$\sum_{i=1}^{k} \sum_{(x,y) \in P^{sol}_i} c(x, y).$$

However, this problem of transcript assembly with paired-end reads becomes NP-hard. We show this by reducing from the NP-hard problem of deciding whether a graph G is 3-colorable, that is, if $V(G)$ can be partitioned into three sets V_A, V_B, V_C such that for no edge of G do both of its endpoints belong to the same set. Such a partition of $V(G)$ is called a *proper coloring* of G.

Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and let $E = \{e_1, \ldots, e_m\}$ be an undirected graph. We create the DAG $P(G)$ drawn in Figure 15.6, which consists of a first stage of n blocks corresponding to the n vertices of G, and a second stage of m blocks corresponding to each edge $e_k = (v_{i_k}, v_{j_k})$ of G, $k \in \{1, \ldots, m\}$. Only some of the vertices and arcs of $P(G)$ have been labeled; when an arc is labeled $[L]$, we mean that in the family of paired subpath constraints we have the constraint $(L, [L])$. Also, we set all costs to 0.

THEOREM 15.4 *The problem of transcript assembly with paired-end reads is NP-hard.*

Proof We show that an undirected graph $G = (V, E)$ is 3-colorable if and only if the DAG $P(G)$ drawn in Figure 15.6 admits a solution to Problem 15.4 with three paths.

For the forward implication, observe that we need at least three paths to solve $P(G)$, since the three arcs v_1, X_1, Y_1 exiting from vertex 0 cannot be covered by the same path, and each of them appears in some constraint. By definition, G is 3-colorable if and only if V can be partitioned into three sets V_A, V_B, V_C such that no arc of G is contained in any of them. We use these three sets to create the three solution paths for the problem of transcript assembly with paired-end reads as follows:

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

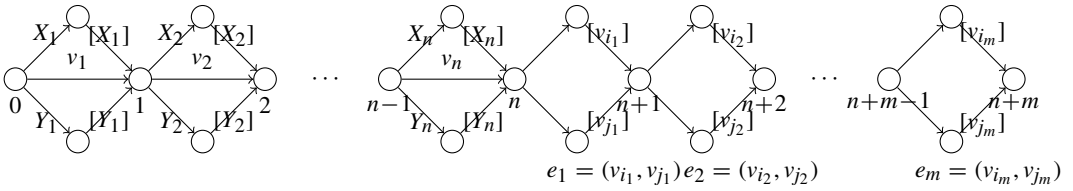


Figure 15.6 A reduction from the graph 3-coloring problem to the problem of transcript assembly with paired-end reads.

- for all $X \in \{A, B, C\}$, in the first stage (until vertex n) path P_X picks up all arcs labeled with a vertex in V_X, and no arc labeled with a vertex in $V \setminus V_X$;
- in the second stage (from vertex n until vertex $n + m$), P_X picks up those arcs labeled with $[v_{i_k}]$ such that v_{i_k} belongs to P_X.

This construction is possible since no edge $e_k = (v_{i_k}, v_{j_k})$ is contained in the same color class, and consequently the two arcs of $P(G)$ labeled v_{i_k} and v_{j_k} do not belong to the same path among $\{P_A, P_B, P_C\}$. Thus, $[v_{i_k}]$ and $[v_{j_k}]$ do not have to be both covered by the same solution path. Therefore, the three paths P_A, P_B, P_C satisfy all path constraints, and they constitute a solution to Problem 15.4.

For the backward implication, suppose that the DAG $P(G)$ drawn in Figure 15.6 admits a solution to Problem 15.4 with three paths P_A, P_B, P_C. Then, we partition V into three color classes A, B, C by setting $v_i \in X$ if and only if the arc of $P(G)$ labeled by v_i (in the first stage from vertex 0 to vertex n) belongs to P_X, for all $X \in \{A, B, C\}$. To see that $\{A, B, C\}$ is indeed a partition of V, observe that in each block k of the first stage of $P(G)$ no two paths in $\{P_A, P_B, P_C\}$ can share an arc, since all three arcs labeled v_k, X_k, Y_k appear in some constraint. Therefore, each arc v_k appears in exactly one of $\{P_A, P_B, P_C\}$. The proof that the partition $\{A, B, C\}$ is also a proper coloring of G follows as in the forward implication. For every arc (v_{i_k}, v_{j_k}) of G, its endpoints belong to different paths among P_A, P_B, P_C, because of the path constraints $(v_{i_k}, [v_{i_k}])$ and $(v_{j_k}, [v_{j_k}])$. □

15.3 Simultaneous assembly and expression estimation

In this section we study the problem in which we need to do simultaneous transcript assembly and expression estimation. We assume that we have already constructed a splicing graph, for example by methods described at the beginning of Section 15.2.

Problem 15.5 Transcript assembly and expression estimation (least squares)

Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, and a weight function $w : E \to \mathbb{Q}_+$, find a collection of paths $P_1, \ldots, P_k$ in G, and their corresponding expression levels $e_1, \ldots, e_k$, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

and the following sum of errors is minimized:

$$\sum_{(x,y)\in E} \left(w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right)^2 . \tag{15.5}$$

The problem formulation we give in Problem 15.5 above coincides with the problem of annotated transcript expression estimation (Problem 15.1), with the amendment that we also need to find the paths starting in some vertex of S and ending in some vertex of T which minimize the least-squared error. Recall that each vertex of the splicing graph is considered as being subdivided as in Section 15.1, so that we have coverages associated only with the arcs of the splicing graph. Observe also that we do not impose an upper bound on the number of paths in a solution. Insight 15.2 discusses some practical strategies for limiting the number of solution paths. See Figure 15.7 for an example.

Problem 15.5 can be solved by a minimum-cost flow problem with convex cost functions (recall Insight 5.2), as follows. Construct the flow network $N = (G^*, \ell, u, c, q)$, where G^* is obtained from G by adding a global source s^* connected to all vertices in S, and a global sink t^* to which all vertices in T are connected. The demand of every arc is set to 0, and the capacity of every arc is unlimited. The cost function for every arc (x, y) of G is $c(x, y, z) = (w(x, y) - z)^2$, which is a convex function in z, and the cost of all arcs incident to s^* or t^* is set to 0. Exercise 15.6 asks the reader to work out the formal details showing that any decomposition into weighted paths of the minimum-cost flow on N gives an optimal solution for Problem 15.5.

We now modify Problem 15.5 into Problem 15.6 by replacing the square function with the absolute value function. See Insight 15.3 for a brief discussion on the differences between the two functions.

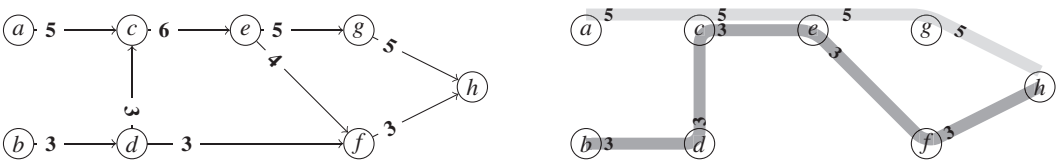


Figure 15.7 On the left, a weighted DAG G for Problem 15.5 or 15.6, with $S = \{a, b\}$ and $T = \{h\}$. On the right, one optimal solution for both problems, consisting of two paths with expression levels 5 and 3, respectively. Their cost is $|6 - 8|^\beta + |3 - 0|^\beta + |4 - 3|^\beta$, for $\beta \in \{1, 2\}$, from arcs (c, e), (d, f), and (e, f).

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Problem 15.6 Transcript assembly and expression estimation (least absolute values)

Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, and a weight function $w : E \to \mathbb{Q}_+$, find a collection of paths $P_1, \ldots, P_k$ in G, and their corresponding expression levels $e_1, \ldots, e_k$, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

and the following sum of errors is minimized:

$$\sum_{(x,y)\in E} \left| w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right|. \tag{15.6}$$

This admits a reduction to a minimum-cost flow problem with linear cost functions (in fact, null or unit costs), which we describe next. Suppose that $P_1, \ldots, P_k$, with expression levels $e_1, \ldots, e_k$, are some paths in G. Let $g : E \to \mathbb{Q}_+$ be the flow on G induced by these paths, namely $g(x,y) = \sum_{j\,:\,(x,y)\in P_j} e_j$. Using g, we can rewrite (15.6), and call it the *error* of g, as

$$\mathsf{error}(g) = \sum_{(x,y)\in E} |w(x,y) - g(x,y)|\,.$$

If $P_1, \ldots, P_k$, with expression levels $e_1, \ldots, e_k$, are optimal for Problem 15.6, then the induced flow g minimizes $\mathsf{error}(g)$. Vice versa, if g is a flow on G minimizing $\mathsf{error}(g)$, then any decomposition of g into weighted paths (obtained by Theorem 5.2) gives an optimal solution to Problem 15.6. Therefore, for solving Problem 15.6, it suffices to find a flow g on G minimizing $\mathsf{error}(g)$ and split it into (an arbitrary number of) weighted paths.

In order to find such an optimal g, we build a flow network N, which we call an *offset network*, such that the flow on an arc (x,y) of N models the quantity $|w(x,y) - g(x,y)|$. In other words, a minimum-cost flow in N models the offsets (positive or negative) between the weights of arcs of G and their expression levels in an optimal solution to Problem 15.6. We achieve this by considering, for every vertex $y \in V(G)$, the quantity

$$\Delta(y) = \sum_{x\in N^+(y)} w(y,x) - \sum_{x\in N^-(y)} w(x,y).$$

If $\Delta(y) > 0$, then we need to force $\Delta(y)$ units of flow to go from vertex y to a global sink t^*. If $\Delta(y) < 0$, then we need to force $-\Delta(y)$ units of flow to go from a global source s^* to vertex y.

Formally, the offset network $N = (G^*, \ell, u, c, q)$ is constructed as follows (see Figure 15.8 for an example).

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

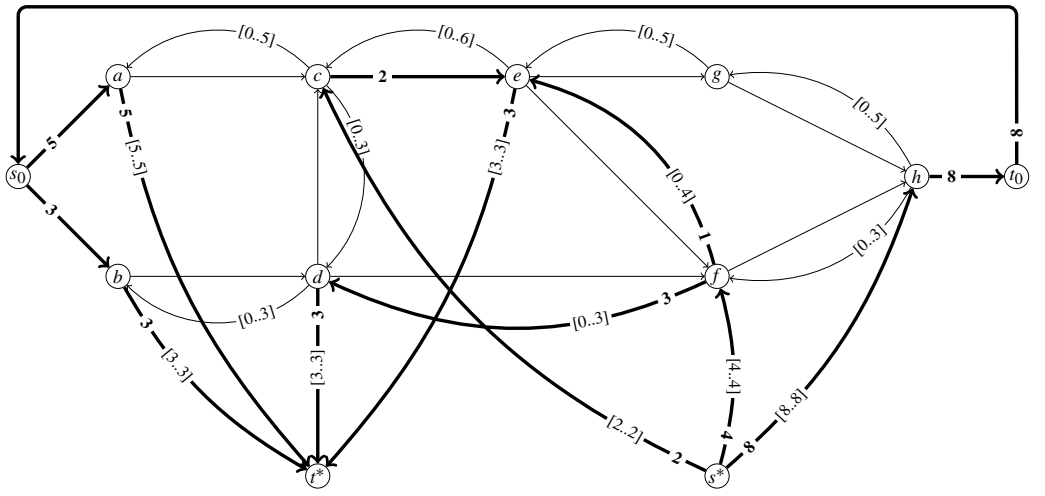


Figure 15.8 The flow network N constructed from the input in Figure 15.7 (left). The arcs between vertices of G have cost 1; all other arcs have cost 0. Arcs with non-null value in a minimum-cost flow f over N are highlighted; f induces the flow drawn in Figure 15.7 (right).

- We add to G^* all vertices and arcs of G, together with
  - a new source s_0 and a new sink t_0, with arcs (s_0, s) for every $s \in S$, arcs (t, t_0), for every $t \in T$, and arc (t_0, s_0), all with infinite capacity, and with null demand and cost;
  - vertices s^* and t^*.
- For every arc $(x, y) \in E(G)$,
  - we set its demand $\ell(x, y) = 0$, its capacity $u(x, y) = \infty$, and its cost $c(x, y) = 1$;
  - we add the reverse arc (y, x) with $\ell(y, x) = 0$, $u(y, x) = w(x, y)$, $c(y, x) = 1$.
- Starting with initial flow value $q := 0$, for every $y \in V(G)$
  - let $\Delta(y) := \sum_{x \in N^+(y)} w(y, x) - \sum_{x \in N^-(y)} w(x, y)$;
  - if $\Delta(y) > 0$ then
    - we add the arc (y, t^*) with $\ell(y, t^*) = u(y, t^*) = \Delta(y)$, $c(y, t^*) = 0$;
    - we update $q := q + \Delta(y)$;
  - if $\Delta(y) < 0$ then
    - we add the arc (s^*, y) with $\ell(s^*, y) = u(s^*, y) = -\Delta(y)$, $c(s^*, y) = 0$.

Given a minimum-cost flow f on N, we construct the desired flow $g : E(G) \to \mathbb{Q}_+$ as follows: given an arc $(x, y) \in E(G)$,

$$g(x, y) := \begin{cases} w(x, y) + f(x, y), & \text{if } f(x, y) > 0, \\ w(x, y) - f(y, x), & \text{if } f(y, x) > 0, \\ w(x, y), & \text{otherwise.} \end{cases} \tag{15.7}$$

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Observe that g is a well-defined function, by the fact that $f(y,x) \leq u(y,x) = w(x,y)$, for every arc $(x,y) \in E(G)$, and by Lemma 15.5 below. Exercise 15.7 asks the reader to show that g is indeed a flow on G. Since, for every arc $(x,y) \in E(G)$, we set $c(x,y) = c(y,x) = 1$ in N, it holds that

$$\mathsf{error}(g) = \mathsf{cost}(f).$$

Thus, since f is of minimum cost, g minimizes $\mathsf{error}(g)$.

LEMMA 15.5 *If f is a minimum-cost flow in N, then at most one of $f(x,y)$ and $f(y,x)$ is non-null, for every arc $(x,y) \in E(G)$.*

Proof We argue by contradiction, and suppose that, for an arc $(x,y) \in E$, we have $\min(f(x,y),f(y,x)) > 0$. Then, we show that we can find a flow f' on N of strictly smaller cost than f, as follows. Flow f' is defined as coinciding with f on all arcs, except for $f'(x,y)$, which equals $f(x,y) - \min(f(x,y),f(y,x))$ and $f'(y,x)$, which equals $f(y,x) - \min(f(x,y),f(y,x))$. Note that f' is a flow on N. Since we assumed $\min(f(x,y),f(y,x)) > 0$, and the cost of both arcs (x,y) and (y,x) is 1, f' has a strictly smaller cost than the minimum-cost flow f. □

Insight 15.2 Limiting the number of transcripts

In Problems 15.5 and 15.6 we did not impose a constraint on the number of paths in an optimal solution. However, in practice, according to the same parsimony principle as was invoked in Section 15.2.1, one prefers a solution with a small number of solution paths.

One way this can be achieved is to split the resulting flow into the minimum number of paths. Since this is an NP-hard problem (Theorem 5.3), one can apply some of the heuristics from Insight 5.1.

However, we can ask for an exact algorithm: Problems 15.5 and 15.6 can be reformulated by receiving in the input also an upper bound on the number k of solution paths. Exercise 15.11 asks the reader to show that both of these problems become NP-hard. Exercise 15.12 asks you to show that these extensions can be solved with an algorithm based on dynamic programming.

Another common practical modification consists of changing the objective function in Problems 15.5 and 15.6 by adding a regularization term $\lambda \sum_{i=1}^{k} e_i$, for some given $\lambda \geq 0$:

$$\sum_{(x,y)\in E} \left(w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right)^2 + \lambda \sum_{i=1}^{k} e_i.$$

Exercise 15.10 asks the reader to modify the minimum-cost flow reductions presented in this section for solving this generalization. Just as before, splitting the resulting flow into the minimum number of paths is also NP-hard, but in practice decompositions of it based on heuristics from Insight 5.1 can lead to solutions with fewer paths.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Therefore, the following theorem holds.

THEOREM 15.6 *The problem of transcript assembly and expression estimation (least absolute values) can be solved in polynomial time.*

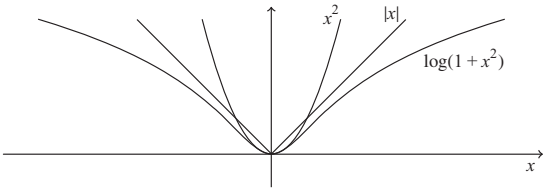
Proof Given an input for the problem of transcript assembly and expression estimation (least absolute values), we construct the flow network N and compute in polynomial time a minimum-cost flow f on N. The flow f induces a flow g on G, defined in (15.7). Using Theorem 5.2 we can split the flow g into weighted paths in time linear in the number of vertices and arcs of the input graph. From the minimality of f, and the fact that $\mathsf{error}(g) = \mathsf{error}(f)$, these paths are optimal for the problem of transcript assembly and expression estimation (least absolute values). □

15.4 Transcript alignment with co-linear chaining

In Section 6.5 we studied a dynamic programming solution for aligning a eukaryote gene sequence to the genome, allowing a limited number of free gaps, modeling the introns. Applying such an algorithm, for example, on long reads obtained by RNA-sequencing is infeasible. For this reason, in Section 10.6 we studied some alternatives using index structures. Now, we are ready for our last proposal for implementing split-read alignment, this time combining maximal exact matches with a rigorous dynamic programming approach. This approach gives a scalable solution to find the path constraints on the splicing graph required in Section 15.2.2.

Insight 15.3 Error functions

Throughout this chapter we have used the sum of squared differences and the sum of absolute differences as objective functions. The absolute value function has the advantage that it is more robust with respect to outliers than the square function. However, in contrast to the square function, it leads to unstable solutions, in the sense that small changes in the coverage values can lead to great differences in the optimal solution. For example, one choice that could mitigate both of these issues is the function $\log(1 + x^2)$. However, this function is not convex, so no network flow solution can apply. Nevertheless, the dynamic programming solution asked for in Exercise 15.13 can apply for this function.



https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

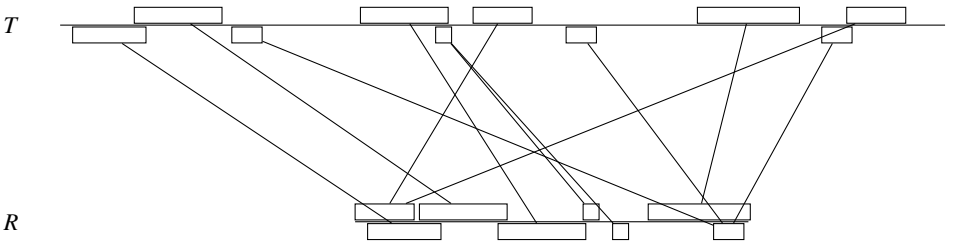


Figure 15.9 Maximal exact matches as input for co-linear chaining for aligning a long read from an RNA transcript to the genome. Boxes cover the area spanned by the matches, and edges show the pairs.

Consider a long read R from an RNA transcript of a gene. Compute the maximal exact matches between R and the genome string T, using the techniques from Section 11.1.3. The alignment result should look like the one illustrated in Figure 15.9.

Co-linear chaining aims to collect a chain of maximal exact matches (or any other local alignment anchors) so that the order of the selected pairs satisfies the order in which they appear in both T and R, and the collected chain covers the maximum area of R, among all such feasible chains. This problem statement is formalized below.

Problem 15.7 Co-linear chaining

Let T and R be two strings, and let V be a set of N tuples (x, y, c, d) with the property that $T_{x..y}$ matches $R_{c..d}$. Find a sequence $S = s_1 s_2 \cdots s_p$ of tuples, where each $s_i \in V$, and

- $s_j.y > s_{j-1}.y$ and $s_j.d > s_{j-1}.d$, for all $1 \leq j \leq p$, and
- the *ordered* coverage of R, defined as

$$\texttt{coverage}(R, S) = |\{i \in [1..|R|] \mid i \in [s_j.c..s_j.d] \text{ for some } 1 \leq j \leq p\}|,$$

  is maximized.

The solution for Problem 15.7 uses dynamic programming and the invariant technique studied in Chapter 6.

First, sort the tuples in V by the coordinate y into the sequence $v_1 v_2 \cdots v_N$. Then, fill a table $C[1..N]$ such that $C[j]$ gives the maximum ordered coverage of $R[1..v_j.d]$ using the tuple v_j and any subset of tuples from $\{v_1, v_2, \ldots, v_{j-1}\}$. Hence, $\max_j C[j]$ gives the total maximum ordered coverage of R. It remains to derive the recurrence for computing $C[j]$. For any tuple $v_{j'}$, with $j' < j$, we consider two cases:

(a) $v_{j'}$ does not overlap v_j in R (that is, $v_{j'}.d < v_j.c$), or
(b) $v_{j'}$ overlaps v_j in R (that is, $v_j.c \leq v_{j'}.d \leq v_j.d$).

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

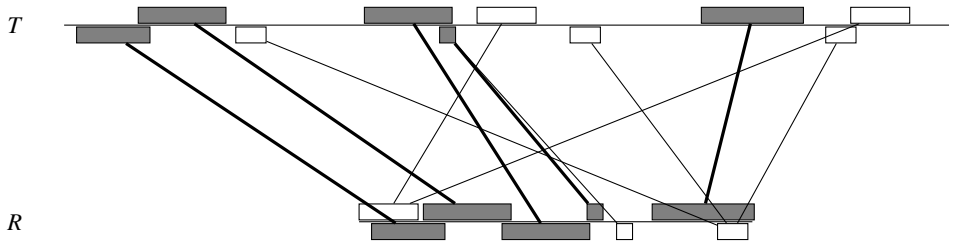


Figure 15.10 Optimal solution of co-linear chaining visualized by the shaded boxes and thicker edges.

The best ordered coverage that uses the tuple v_j and a subset of tuples from $\{v_1, v_2, \ldots, v_{j-1}\}$ with none of them overlapping v_j (case (a) above) can thus be obtained by the recurrence

$$C^{\mathrm{a}}[j] = \max_{j' \,:\, v_{j'}.d < v_j.c} C[j'] + (v_j.d - v_j.c + 1). \tag{15.8}$$

Similarly, the best ordered coverage that uses the tuple v_j and a subset of tuples from $\{v_1, v_2, \ldots, v_{j-1}\}$ with at least one of them overlapping v_j (case (b) above) can be obtained by the recurrence

$$C^{\mathrm{b}}[j] = \max_{j' \,:\, v_j.c \le v_{j'}.d \le v_j.d} C[j'] + (v_j.d - v_{j'}.d), \tag{15.9}$$

which works correctly unless there is a tuple $v_{j'}$ satisfying the condition $v_j.d \ge v_{j'}.d \ge v_{j'}.c > v_j.c$. Such containments can, however, be ignored when computing the final value $C[j] = \max(C^{\mathrm{a}}[j], C^{\mathrm{b}}[j])$, because case (a) always gives a better result: see Exercise 15.16.

We can use the invariant technique to implement these recurrence relations with range maximum queries. These queries can be solved using the search tree in Lemma 3.1 on page 20 (in fact, here we need its dual version, with minimum replaced by maximum, which we denote as RMaxQ):

$$\begin{aligned}
C^{\mathrm{a}}[j] &= (v_j.d - v_j.c + 1) + \max_{j' \,:\, v_{j'}.d < v_j.c} C[j'] \\
&= (v_j.d - v_j.c + 1) + \mathcal{T}.\mathsf{RMaxQ}(0, v_j.c - 1), \\
C^{\mathrm{b}}[j] &= v_j.d + \max_{j' \,:\, v_j.c \le v_{j'}.d \le v_j.d} C[j'] - v_{j'}.d \\
&= v_j.d + \mathcal{I}.\mathsf{RMaxQ}(v_j.c, v_j.d), \\
C[j] &= \max(C^{\mathrm{a}}[j], C^{\mathrm{b}}[j]).
\end{aligned}$$

For these to work correctly, we need to have properly updated the trees $\mathcal{T}$ and $\mathcal{I}$ for all $j' \in [1..j-1]$. That is, we need to call $\mathcal{T}$.`Update`$(v_{j'}.d, C[j'])$ and $\mathcal{I}$.`Update`$(v_{j'}.d, C[j'] - v_{j'}.d)$ after computing each $C[j']$. The running time is $O(N \log N)$. The pseudocode is given in Algorithm 15.1.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

Algorithm 15.1: Co-linear chaining

Input: A set of tuples V sorted by y-coordinate into sequence $v_1 v_2 \cdots v_N$.
Output: The index j giving $\max_j C[j]$.

Initialize search trees $\mathcal{T}$ and $\mathcal{I}$ with keys $v_j.d$, $1 \leq j \leq N$, and with key 0, all keys associated with values $-\infty$;
$\mathcal{T}$.`Update`$(0, 0)$;
$\mathcal{I}$.`Update`$(0, 0)$;
for $j \leftarrow 1$ *to* N **do**
  $C^{\mathrm{a}}[j] \leftarrow (v_j.d - v_j.c + 1) + \mathcal{T}.\mathsf{RMaxQ}(0, v_j.c - 1)$;
  $C^{\mathrm{b}}[j] \leftarrow v_j.d + \mathcal{I}.\mathsf{RMaxQ}(v_j.c, v_j.d)$;
  $C[j] \leftarrow \max(C^{\mathrm{a}}[j], C^{\mathrm{b}}[j])$;
  $\mathcal{T}$.`Update`$(v_j.d, C[j])$;
  $\mathcal{I}$.`Update`$(v_j.d, C[j] - v_j.d)$;
return $\operatorname{argmax}_j C[j]$;

Figure 15.10 illustrates the optimal chain on our schematic example. This chain can be extracted by modifying Algorithm 15.1 to store traceback pointers.

THEOREM 15.7 *Co-linear chaining (Problem 15.7) on N input tuples can be solved in $O(N \log N)$ time.*

15.5 Literature

The least-squares method given in Section 15.1 is a standard topic in numerical analysis and statistics (it is attributed to Gauss in 1795). Faster solutions are possible if approximation is allowed and the number of equations is much larger than the number of variables (Drineas *et al.* 2011).

A very similar minimum path cover formulation to the problem of transcript assembly with short reads from Section 15.2.1 was introduced by Trapnell *et al.* (2010). The differences are that Trapnell *et al.* (2010) uses an overlap graph (which is extended to paired-end reads, for a suitably defined notion of overlap), and the costs are assigned only to the edges of the bipartite graph arising from the alternative solution of the minimum path cover problem via a bipartite matching problem (Fulkerson 1956). An even earlier connection between the minimum path cover problem and a transcript assembly problem is considered in Jenkins *et al.* (2006). The problem of transcript assembly with long reads from Section 15.2.2 was initially proposed in the context of transcript assembly by Bao *et al.* (2013), and fully solved in Rizzi *et al.* (2014). Exercise 15.3 asks the reader to rediscover another reduction, from Ntafos & Hakimi (1979), of the unweighted version of this problem to a minimum-cost flow problem.

The problem of transcript assembly with paired-end reads from Section 15.2.3 was proposed in Song & Florea (2013). The unweighted versions of these two problems first appeared in an application to program testing (Ntafos & Hakimi 1979). The latter was

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

proved NP-hard in Ntafos & Hakimi (1979) by a reduction from 3-SAT and in Beerenwinkel *et al.* (2014) and Rizzi *et al.* (2014) by a reduction from graph 3-colorability.

The problem of transcript assembly and expression estimation (least squares) from Section 15.3 was considered in Tomescu *et al.* (2013a), and variants of it appeared before in Li *et al.* (2011a) and Feng *et al.* (2011). Its least absolute values variant appeared in Lin *et al.* (2012), and the solution we presented is adapted from the one given in Tomescu *et al.* (2013a).

The addition of a regularization term mentioned in Insight 15.2 was considered by Li *et al.* (2011b) and Bernard *et al.* (2014). The addition of an input upper bound on the number of predicted transcripts was proposed by Tomescu *et al.* (2013b).

The co-linear chaining solution is analogous to one in Abouelhoda (2007). Exercise 15.17 covers some variants studied in Mäkinen *et al.* (2012).

Exercises

15.1 Work out the calculations for deriving the system of equations (15.3), using the formula

$$\begin{aligned}\frac{\partial \left(w_i - \alpha_{i,1}e_1 - \alpha_{i,2}e_2 - \cdots - \alpha_{i,k}e_k\right)^2}{\partial e_j} &= 2\left(w_i - \alpha_{i,1}e_1 - \alpha_{i,2}e_2 - \cdots - \alpha_{i,k}e_k\right) \\ &\quad \cdot \frac{\partial \left(w_i - \alpha_{i,1}e_1 - \alpha_{i,2}e_2 - \cdots - \alpha_{i,k}e_k\right)}{\partial e_j} \\ &= -2\alpha_{i,j}\left(w_i - \alpha_{i,1}e_1 - \alpha_{i,2}e_2 - \cdots - \alpha_{i,k}e_k\right).\end{aligned}$$

15.2 Formalize the exon detection phase described on page 329 using HMMs (recall Insight 14.1 and Chapter 7).

15.3 Consider the following version of Problem 15.3 in which there are no costs associated with the arcs for the DAG and in which we need to cover only a given collection of paths. Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, and a collection $\mathcal{P}^{in} = \{P_1^{in}, \ldots, P_t^{in}\}$ of directed paths in G, find the minimum number k of paths $P_1^{sol}, \ldots, P_k^{sol}$ such that

- every P_i^{sol} starts in some vertex of S and ends in some vertex of T,
- every path $P^{in} \in \mathcal{P}^{in}$ is entirely contained in some P_i^{sol}.

Assume also that no P_i^{in} is entirely included in another P_j^{in} (otherwise P_i^{in} can be removed from $\mathcal{P}^{in}$). Show that this problem can be solved by a reduction to a minimum-cost flow problem (without having to iteratively merge paths with longest suffix–prefix overlaps). What features does the resulting flow network have? Can you apply a specialized minimum-cost flow algorithm with a better complexity for such a network?

15.4 Given an input for the problem of transcript assembly with paired-end reads, show that we can decide in polynomial time whether it admits a solution with two paths, and if so, find the two solution paths.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

15.5 Using the reduction in the proof of Theorem 15.4, conclude that there exists for no $\varepsilon > 0$ an algorithm returning a solution with k paths for the problem of transcript assembly with paired-end reads, where k is greater than the optimal number of paths by a multiplicative factor $\frac{4}{3} - \varepsilon$.

15.6 Argue that the reduction of Problem 15.5 to a network flow problem with convex costs presented on page 338 is correct.

15.7 Show that the function g defined in (15.7) on page 340 is a flow on G, namely that

- for all $y \in V(G) \setminus (S \cup T)$, $\sum_{x \in N^-(y)} g(x,y) = \sum_{x \in N^+(y)} g(y,x)$;
- $\sum_{s \in S} \sum_{y \in N^+(s)} g(s,y) = \sum_{t \in T} \sum_{x \in N^-(t)} g(x,t)$.

15.8 Explain what changes need to be made to the flow network N, if in Problem 15.6 we also get in the input a coefficient $\alpha(x,y)$ for every arc (x,y) of G, and we are asked to find the paths and their expression levels which minimize

$$\sum_{(x,y) \in E} \alpha(x,y) \left| w(x,y) - \sum_{j \,:\, (x,y) \in P_j} e_j \right| .$$

15.9 Consider a variant of Problem 15.6 in which we want to minimize the absolute differences between the total coverage (instead of the average coverage) of an exon and its predicted coverage. Explain how this problem can be reduced to the one in Exercise 15.8 above, for an appropriately chosen function α.

15.10 Adapt the reduction to a minimum-cost flow problem from Section 15.3 to solve the following problem. Given a DAG $G = (V,E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, and a weight function $w : E \to \mathbb{Q}_+$, find a collection of paths $P_1, \ldots, P_k$ in G, and their corresponding expression levels $e_1, \ldots, e_k$, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

and the following function is minimized:

$$\sum_{(x,y) \in E} \left| w(x,y) - \sum_{j \,:\, (x,y) \in P_j} e_j \right| + \lambda \sum_{i=1}^{k} e_i .$$

What can you say about the problem in which we use the squared difference, instead of the absolute value of the difference?

15.11 Show that the following problem is NP-hard, for any fixed $\beta \geq 1$. Given a DAG $G = (V,E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, a weight function $w : E \to \mathbb{Q}_+$, and an integer k, find a collection of paths $P_1, \ldots, P_k$ in G, and their corresponding expression levels $e_1, \ldots, e_k$, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

and the following function is minimized:

$$\sum_{(x,y)\in E} \left| w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right|^{\beta}.$$

Hint. Use Theorem 5.3 or its proof.

15.12 Show that, given $k \geq 1$ and given $e_1, \ldots, e_k$, the following problem is solvable in time $O(n^{2k+2})$. Given a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, a weight function $w : E \to \mathbb{Q}_+$, and an integer k, find a collection of paths $P_1, \ldots, P_k$ in G, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

and the following function is minimized:

$$\sum_{(x,y)\in E} \left| w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right|. \tag{15.10}$$

Hint. Use dynamic programming and consider the optimal k paths ending in every tuple of k vertices.

15.13 Show that, given an error function $\delta : \mathbb{Q}_+ \times \mathbb{Q}_+ \to \mathbb{Q}_+$, the problem considered in Exercise 15.12 in which the objective function (15.10) is replaced with

$$\sum_{(x,y)\in E} \delta\left(w(x,y), \sum_{j\,:\,(x,y)\in P_j} e_j \right) \tag{15.11}$$

is solvable with the same time complexity $O(n^{2k+2})$ (assuming δ is computable in time $O(1)$).

15.14 Consider the following problem of *transcript assembly and expression estimation with outliers*, in which the weights of the arcs not belonging to any solution path do not contribute in the objective function. Given $k \geq 1$, a DAG $G = (V, E)$, a set $S \subseteq V$ of possible start vertices, a set $T \subseteq V$ of possible end vertices, a weight function $w : E \to \mathbb{Q}_+$, and an integer k, find a collection of paths $P_1, \ldots, P_k$ in G, such that

- every P_i starts in some vertex of S and ends in some vertex of T,

and the following function is minimized:

$$\sum_{(x,y)\text{: exists } P_i \text{ with } (x,y)\in P_i} \left| w(x,y) - \sum_{j\,:\,(x,y)\in P_j} e_j \right|.$$

Show that this problem is NP-hard. *Hint.* Use Theorem 5.3 or its proof.

15.15 Reduce the problem of transcript assembly and expression estimation with outliers from Exercise 15.14 to the generalized problem from Exercise 15.13, by showing an appropriate error function δ.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press

15.16 Prove that the co-linear chaining algorithm works correctly even when there are tuples containing other tuples in T or in R, that is, tuples of type (x, y, c, d) and (x', y', c', d') such that either $x < x' \leq y' < y$ or $c < c' \leq d' < d$ (or both).

15.17 Modify the co-linear chaining algorithm to solve the following variations of the ordered coverage problem.

(a) Find the maximum ordered coverage of R such that all the tuples involved in the coverage must overlap in R.
(b) Find the maximum ordered coverage of R such that the distance in R between two consecutive tuples involved in the coverage is at most a given threshold value α.
(c) Find the maximum ordered coverage of R such that the distance in T between two consecutive tuples involved in the coverage is at most a given threshold value β.

15.18 Co-linear chaining gives a rough alignment between the transcript and the genome as a list of subregion correspondences. Consider how this rough alignment can be fine-grained into a sequence-level alignment.

https://doi.org/10.1017/CBO9781139940023.017 Published online by Cambridge University Press