Exploring the Landscape of Transcripts

Thomas Gatter Jörg Fallmann Peter F. Stadler

April 22, 2022

1 Introduction

Eukaryotic transcripts do not derive as direct copies of DNA regions. Primary transcripts are transcribed without changes from their respective template DNA but are further processed to mature RNA. Active regions called *exons* are interrupted by *introns*. Introns are excised and the respective ends of the neighboring exons ligated to close the nascent gaps in a process called *splicing* (see also Fig. 1). If multiple possible arrangements of exons exist for one gene, we say variation derives of *alternative splicing* (AS), and each variant mature transcript constitutes an *isoform*. The splicing process is non-random and well regulated by complex 'splicing programs'. Commonly, (sub-)regions may be both intronic and exonic, meaning they can be both spliced out or retained depending on a respective isoform. Examples for AS include alternative start- or end-sites, extending, shortening, skipping, or including exons, or retaining intron sequences (see Fig. 2). AS has been established as a key factor in cell regulation, e.g. in gene regulation in embryonics or diseases.

Eukaryotic transcriptomes exhibit a high degree of diversity that is not limited to higher Eukaryotes. Genome-wide studies on human tissue revealed that 95% of protein coding multi-exon genes and 30% of non-coding RNAs undergo alternative splicing. Similar ratios have been found also for non-human targets, such as mice.

While the abundance of DNA remains constant[1] – and errors in coverage uniformity only arise within the execution of the sequencing protocol – the abundance of RNA transcripts varies naturally. Differences in the abundance are both diverse and often substantial, even between isoforms of the same gene. Transcription and arising copy numbers of isoforms are governed by a complex set of regulatory elements and conditional among others to environmental stress, developmental status, celltypes, or illness. Actively transcribed genes are said to be *expressed* at the level of their abundance. Lowly expressed transcripts may not be captured by a typical sequencing setup, relating to the chosen sequencing depth. Meta studies of RNA-seq experiments indicate that many rare isoforms have evaded annotation, and typically sized RNA-seq experiments miss out significant portions of low abundant spliceforms.

Under ideal conditions sequencing protocols should only produce reads from mature transcripts. However, contamination with incompletely spliced, un-mature RNA is common and problematic. True intron retention events in particular may be indistinguishable to such noise. Furthermore, splicing errors and other aberrant products of the transcription process may create misleading, but e.g. due to maladaption consistent, signals. The generally high number of very low abundant spliceforms has been attributed to such effects, disputing their functional significance. Yet, aberrant splicing has been established as a key factor in diseases including cancer.

Next to typical, and especially cheap, *single-end* sequencing where only one read is produced per DNA (or to cDNA translated RNA) template, nearly all platforms also allow *paired-end* sequencing. Reads here are created in strict pairs, as the name suggest, with reads originating from opposite ends and strands of the DNA template (Fig. 3). The connecting region between read-pairs remains unsequenced, but the approximate distance between them, the *insert size*, is known. Paired-end reads are indispensable for assembly, owing to the observation that pairs by construction originate in close proximity and from the same genomic region.

2 Transcript Assembly

Compared to previous attempts to model genes *de novo* based on signals in the genome sequence in terms of coding regions and splice sites, RNA sequencing (RNA-seq) offers a much more direct and nuanced view of the

[1] Constant here is used in the sense that the amount of available DNA for sequencing does not vary depending on the genomic position or feature. Reads are accordingly produced uniformly over all genomic positions. Based on duplications and deletions within the genome some sequence motifs will appear to have higher or lower coverage, although regions have the same base coverage.

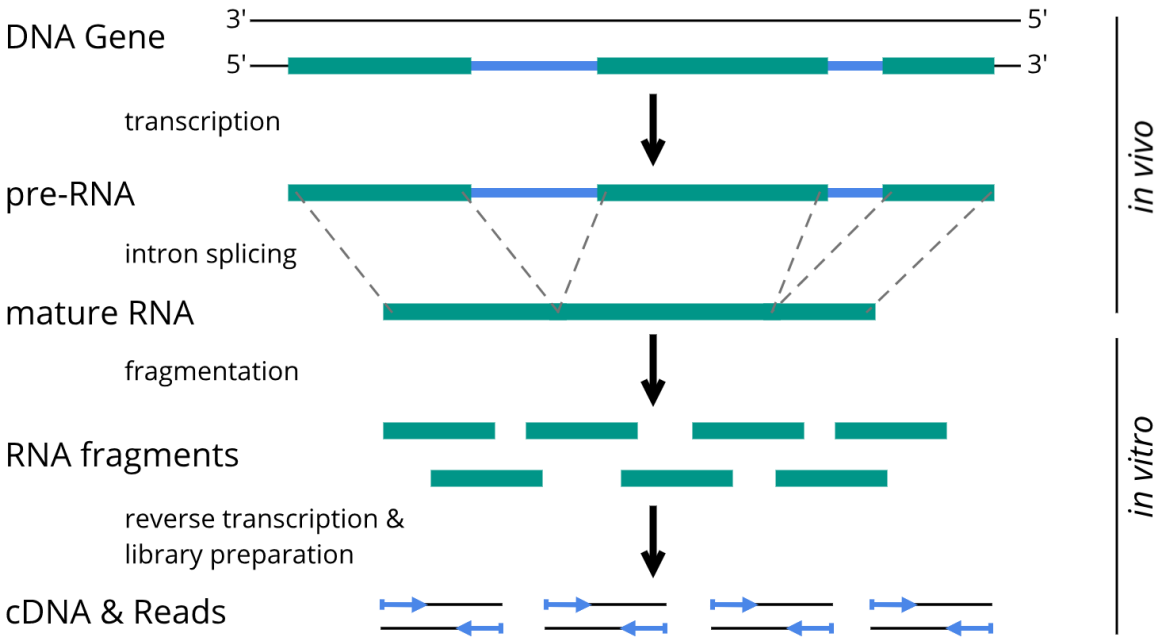

Figure 1: Schematic overview of RNA-Seq. A gene is transcribed, including introns, from a single strand of double stranded DNA to pre-RNA. Introns are removed – spliced – to derive mature RNA. Mature RNA is extracted from the cell and fragmented before reverse transcription to cDNA. The resulting cDNA framents are sequenced like genomic DNA would, e.g. by a paired-end sequencing.

actual structure of transcribed products and how they are processed. Regardless, the accurate classification of the transcriptational output remains a very challenging task.

If a reliable genome assembly is available, the RNA-seq reads are aligned against this reference by specialized state-of-the-art split alignment tools (e.g. `TopHat2`, `STAR`, or `HISAT2`). This alignment is then used as a basis of all further computations. Reference based methods are generally preferable to pure *de novo* methods which are less accurate and computationally more complex.

Transcript assembly encompasses two interdependent tasks: (i) Finding the set of expressed isoforms and (ii) their expression levels. Neither can be observed directly. Instead, reads are composed of a mixture of superimposed isoforms.

Mapping split-reads against a reference results in a set of intervals in genomic coordinates whereby each read provides evidence for (partial) segments of one or more exons. Specifically, exons are identified as consecutive stretches of mapped regions, with splices, i.e. gaps between the intervals of a read, indicating introns. Measuring single transcripts directly is not possible in general due to superimposing features of transcripts, i.e. shared exons and introns. The coverage of exons and splice-sites derived from alignments thus constitute the sum of all transcripts sharing the respective feature.

Graphs are not based on read sequences or k-mers thereof, but rather utilize the set of established exons and introns. This strategy enables a significant reduction with regard to complexity and memory requirements. As a universal concept, overlapping, alternative isoforms are represented as a graph structure where each isoform represents a unique path.

Existing (reference-based) assemblers are strongly limited by the expressive power of short read sequences. Contemporary methods near exclusively rely on only three properties of short reads towards this goal: (1) Reads form nodes, edges or paths, depending on the underlying formalism. Reads spanning two or more exons thereby relay the connectivity of exons and induce diverging paths for alternative splicing events. (2) Paired-end reads yield additional data on the connectivity between more distant exons. (3) The coverage induced by reads is annotated on nodes and/or edges. We refer to the set of (partial) exons a read is composed of as a bin.

There is no consensus on an optimal design for assembly graphs, and methods implement diverse datastructures (see Fig. 4). In this course we want to consider mainly basic splice (or connectivity) graphs. Here, full or partial exons are the nodes, and edges represent either splices or neighboring partial exons. Transcripts are

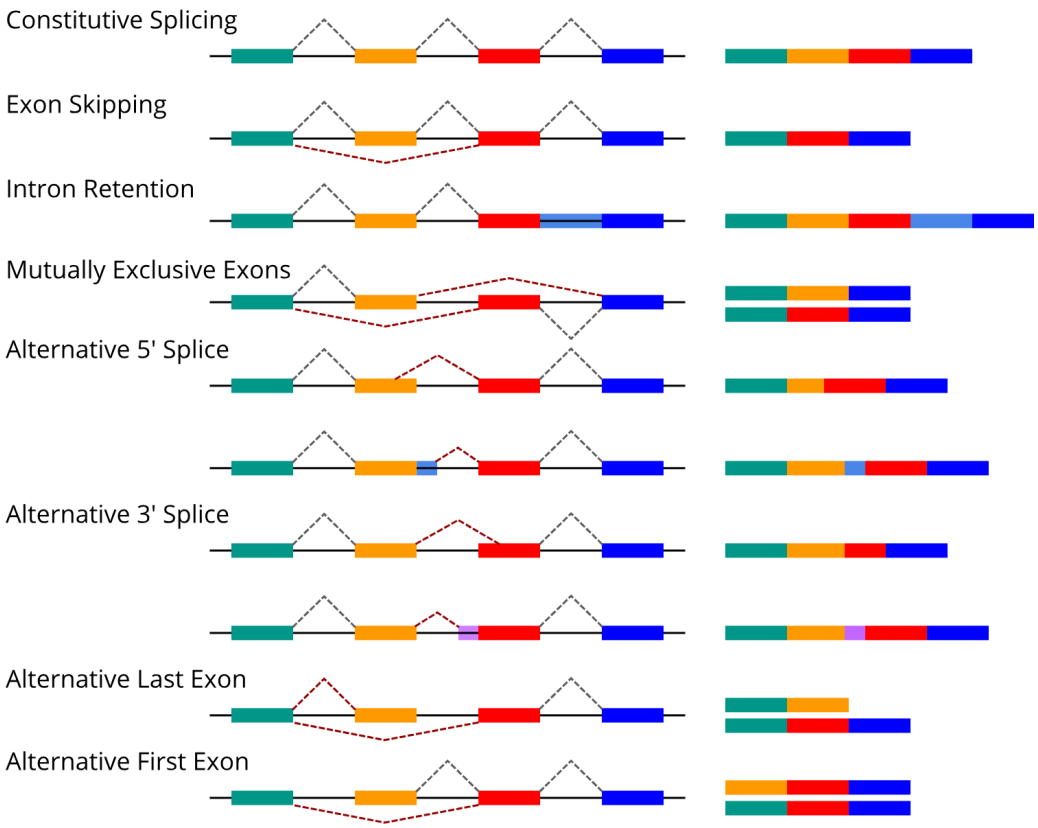

Figure 2: Examples of prevalent alternative splicing categories. Exons are represented by colored boxes and introns by a black line. Splices are indicated by dashed lines at the top or bottom of each exon chain. Black and red lines constitutive alternative splicing events. The final products after splicing are represented to the right. Multiple categories of alternative splicing may be present in the same isoform and different isoforms on the same set of exons exhibiting different categories of alternative splicing often overlap in realistic genes.

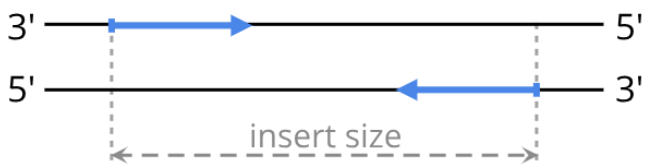

Figure 3: During paired-end sequencing reads are produced from both strands of the fragment in opposite directions. Fragment length determines the distance between both reads, we say their insert size. The unsequenced region between a read-pair is affected by read length and insert size.

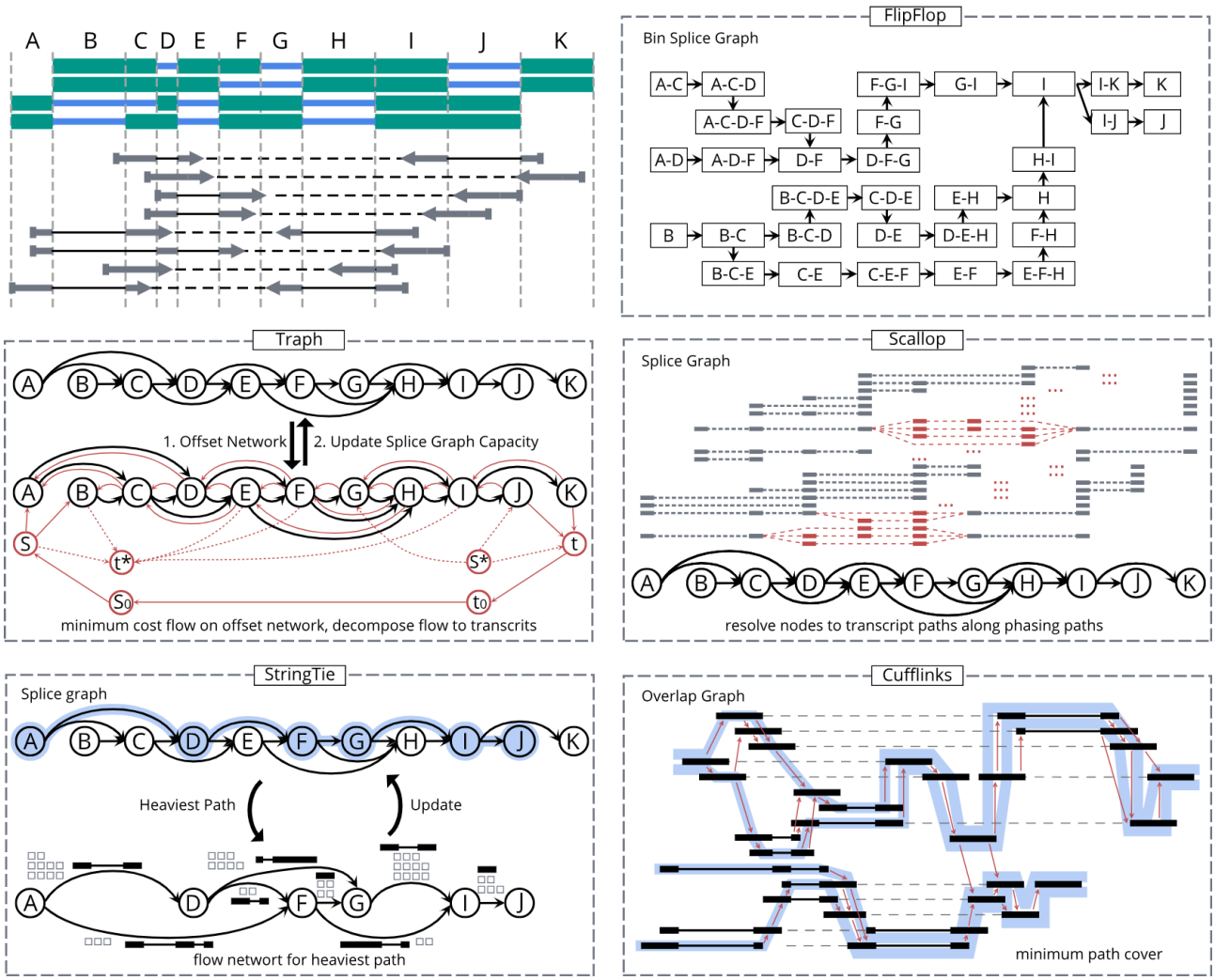

Figure 4: Overview of core datastructures of the most popular tools for transcript assembly. Core datastructures are illustrated using the same set of isoforms (Top Left) on a shared gene. (Partial) Exons are identified by Letters A to K. A set of potential paired reads as depicted. All methods start with a set of RNA-seq reads that have been mapped to a reference and been reduced to a set of exon and intron borders. All assemble transcripts as paths in the respective graph structure. Tools utilize vastly different methods and graphs. `FlipFlop` uses a unique bin-graph design. A penalized maximum likelihood problem is solved to find optimal abundances and their decomposition to a set of transcripts. `Traph`, `Scallop`, and `StringTie` all use basic splice graphs. `Traph` does not utilize multi-splice or paired-end reads. A min-cost flow is determined in an offset network to denoise abundances prior to graph decomposition. `Scallop` converts multi-splice and paired-end reads to a set of paths in the assembly graph to retain their connecting information. If the unknown context between two paired reads is not unique, all possible paths closing this gap are added with equal significance. Nodes are resolved to a set of transcripts constrained to these paths. `Stringtie` iteratively extracts the heaviest path compatible to multi-splice and paired-end reads, constructs a flow network to estimate abundance, and then updates the splice graph by removing reads associated with the flow. `Cufflinks` uses an overlap graph where each read is presented as a node. Nodes are connected if reads overlap and potential paired partners are consistent. Transcripts are extracted as a minimum path cover.

represented as paths in the graph. Observed coverage can be uniquely assigned to each node (exon) and edge (splice-site). Bins containing more than two exons define paths in the graph. Commonly, artificial source s and drain t nodes are added such that all full transcript paths start and end at the same nodes.

Finding a set of transcripts, even in perfected data, is framed as the application of two nested NP-hard problems. Extracting a minimal set of transcript paths with minimal deviation to observed coverage, e.g. modeled as the decomposition of a network flow or an optimization problem as we will see later, is NP-Hard. Finding viable transcript paths is equivalent to finding a superwalk including all read paths (consistent to bins and paired reads), an NP-hard problem well studied for de Bruijn graph assembly.

Naturally, exact solutions are intractable in practice. High noise levels in observed coverage further complicate models, both from a theoretical and practical perspective.

3 Exploring the Transcript-Space

Formalizing the definitions of the last section, we seek the set of expressed isoforms $T = (t_1, \ldots, t_N)$, here of size N, where each t_i represents a sequence of exons and splice-junctions. Additionally we seek the expression levels $F = (f_1, \ldots, f_N)$ of every transcript as real values $f_i \in \mathcal{R}$.

Again, this data cannot be directly observed. Hence we measure only local coverage: Let $E = (e_1, \ldots, e_K)$ be the ordered set of K (sub)-exons of lengths $L = (l_1, \ldots, l_K)$. Analogously, let $S = (s_1, \ldots, s_Q)$ be the ordered set of Q splice-junctions.

In order to progress we require a mapping between exons and splice-sites to (potential) transcripts. We define incidence matrices $A_E = (\alpha^E_{ij})_{N\times K}$, where $\alpha^E_{ij} = 1$ iff the i-th isoform contains the j-th (sub)-exon and $A_S = (\alpha^S_{ij})_{N\times Q}$, where $\alpha^S_{ij} = 1$ iff the i-th isoform contains the j-th junction. The expected number of nucleotides covered by fragments in the i-th exon can be expressed as $l_i \sum_{j=1}^{N} \alpha^E_{ji} f_j$. Likewise the expected number of reads covering the i-th splice junction is $\sum_{j=1}^{N} \alpha^S_{ji} f_j$. In perfect data, observed coverage for either type of feature equals exactly these sums. In real data, signal noise obscures this relationship, suggesting optimization strategies minimizing the local differences between observation and prediction.

One therefore defines a further set of incidence matrices encoding the relationship between fragments and exons or junctions, respectively. Assume M fragments are observed, we define $\mathbf{\Gamma}_E = (\gamma^E_{ij})_{M\times K}$ where γ^E_{ij} equals the number of nucleotides the i-th fragment overlaps to the j-th (sub-)exon. Analogously we define junction incidences $\mathbf{\Gamma}_S = (\gamma^S_{ij})_{M\times Q}$ where $\gamma_{ij} = 1$ iff the i-th fragment overlaps to the j-th junction. Assuming a plausible set of possible isoforms can be established, their expressions may be optimized as

$$\arg\min_F \ \mathbf{sum\_error}(F, A, L, \Gamma) = \arg\min_F \ \sum_{j=1}^{K} \left\| \sum_{i=0}^{M} \frac{\gamma^E_{ij}}{l_j} - \sum_{i=0}^{N} \alpha^E_{ij} f_i \right\| + \sum_{j=1}^{Q} \left\| \sum_{i=0}^{M} \gamma^S_{ij} - \sum_{i=0}^{N} \alpha^S_{ij} f_i \right\|.$$

Since expression levels are non-negative, $f_i \geq 0$ for all $1 \leq i \leq N$. Linear or Quadratic Programming may be used to solve this optimization problem, depending on the choice of the norm $\|\cdot\|$. A sparsity constraint may be added to minimize the number of expressed isoforms. We express this as an application of the L0 norm on the expression vector

$$\|F\|_0 = \sum_{i=0}^{N} \mathrm{card}(f_i) \text{ where } \mathrm{card}(x) = \begin{cases} 0 & x == 0 \\ 1 & x > 0 \end{cases}$$

There are two equivalent and common methods to integrate the sparsity constraint in the optimization. As a first option we may add it as a weighted factor to the optimization function its self. The optimization thus transforms to:

$$\arg\min_F \ \mathbf{sum\_error}(F, A, L, \Gamma) + \lambda \|F\|_0 \quad \textbf{s.t.} \quad \forall f \in F : f \geq 0$$

Alternatively, the sparsity constraint is directly added to the constraint set.

$$\arg\min_F \ \mathbf{sum\_error}(F, A, L, \Gamma) \quad \textbf{s.t.} \quad \forall f \in F : f \geq 0, \|F\|_0 \leq \mu$$

Both optimal λ and μ have to be experimentally determined. A sparse solution will in general exhibit a higher error to the observation than a less sparse one. A suitable trade-off between accuracy and sparseness needs to be established.

As the L0 sparsity constraint is particularly hard to solve in practice, the L1 norm is more commonly implemented as a heuristic. We then optimize

$$\arg\min_{F} \ \textbf{sum\_error}(F, A, L, \Gamma) \quad \textbf{s.t.} \quad \forall f \in F : f \geq 0, \sum_{i=0}^{N} f_i \leq \mu$$

Establishing a viable set of transcripts constitutes a mayor bottleneck for this type of optimization. An *a priori* enumeration of all plausible transcripts has proven to be intractable for complex genes. Even for medium sized loci an overestimation of plausible transcripts causes mayor increases in runtimes. The optimization itself disregards read pairs and bins as a direct influence. However, they define strict constraints on the plausibility of transcripts. Accordingly they restrict the space of eligible paths, yielding a straight forward truncation mechanism for the enumeration of paths. Per definition, splice graphs are acyclic and all transcript paths start and end at common nodes, respectively s and t. A full path enumeration is achieved by simple recursion as follows:

```
1: procedure PathEnumeration(Node v, Path p)
2:     if v == t then
3:         Output p
4:         return
5:     end if
6:     for each successor u of v do
7:         PathEnumeration(u, p + [u])
8:     end for
9: end procedure
10: PathEnumeration(s, [s])
```

Paths incompatible to encountered bins may be abandoned and do not need to be enumerated. We introduce a further parameter to the recursion storing the set of active bins. When following an edge/visiting the next node:

- remove all active bins incompatible to this edge
- if the set is now empty but was not upon entering this node, stop execution of this path
- if not add all new implied bins to the set
- remove inactive bins, i.e. bins for which all exons have been visited

Accordingly, at each iteration, only edges compatible to the active bin set can be followed. The integration of paired-end data is more challenging, as we cannot know when the second partner should have been encountered after finding the first. We have two options. (a) Pairs are approximated as a set of bins. All possible paths connecting both partners in the splice graph are enumerated. A new bin is added per path including the exon sets of both pairs and the respective path. (b) Paired reads are applied as a post-filter to the set of paths. We group pairs with the identical first partner (bin). Then for each group remove any path matching the common bin but none of the second partners. Optionally read pairs where both bins are a the subset of another pair may be omitted to increase specificity for either option.

Representing read data in the form of a splice graph does not only allow us the straight forward enumeration of transcripts, but also a more effective representation of the optimization problem. From hereon, we consider a special form of splice graphs to simplify notation. Instead of full exons, we now designate nodes in the graph to indicate exon borders. Accordingly, we split each node in the traditional splice graph into two new nodes v_{in}, v_{out} with an edge $v_{in} \rightarrow v_{out}$ connecting them. The number of reads aligned to each exon normalized by exon length is labeled on this edge instead of the former node. Edges of splice junctions are labeled by the number of reads indicating the junction as before. We define a common coverage function as follows:

$$\text{cov}(x) = \begin{cases} \sum_{i=0}^{M} \frac{\gamma_{ij}^{E}}{l_j} & x \text{ defines the j-th exon} \\ \sum_{i=0}^{M} \gamma_{ij}^{S} & x \text{ defines the j-th spice-junction} \end{cases}$$

To this effect, now both splice-junctions and exons are represented by to each other consistent edges. Based on this we may redefine and simplify the above optimization. Again, we seek the set of expressed isoforms T,

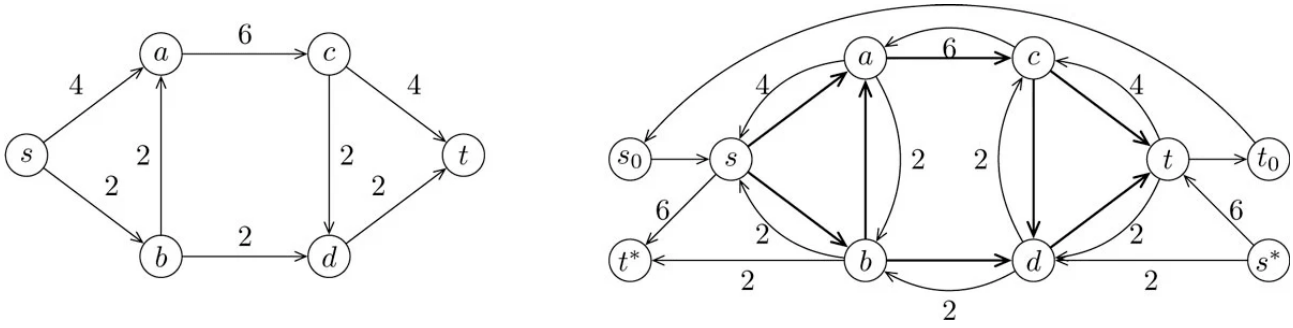

Figure 5: Example of an offset network. An input Graph (left) and the corresponding offset network. Arcs are labeled with their capacity. Unlabeled arcs having infinite capacity.

now defined as paths in the splice graph G, and their expression levels F. We define a new combined incidence matrices $A = (\alpha_{ij})_{N \times K+Q}$, where $\alpha_{ij} = 1$ iff the path of the i-th isoform contains the j-th edge.

$$\arg\min_F \ \textbf{sum\_error}(F, A, G) = \arg\min_F \ \sum_{j=1}^{Q+K} \left\| \text{cov}\,(e_j) - \sum_{i=0}^{N} \alpha_{ij} f_i \right\| .$$

Sparsity constraints may be added in identical fashion as before.

Tomescu *et al.* for the tool `Traph` showed that the minimization problem may also be solved as a min-cost network flow on a transformation of the splice graph. Notably, here the minimization of the error between observed and expected coverage can be computed without an explicit list of transcripts, the network flow condition acting implicitly as a substitute. Any decomposition of the network flow into a set of paths then constitutes an optimal solution, albeit without a sparsity constraint. An optimal decomposition into a minimal set of paths remains NP-hard, but different efficient heuristics become available.

Given a splice graph G we define its offset graph G^* with an edge cost function c as follows (see Fig. 5 for an example):

1. add to G^* the source s and drain t of G, together with
   - new nodes s_0 and t_0
   - edges $s_0 \to s$, $t \to t_0$, and $t_0 \to s_0$, each at infinite capacity and 0 cost
   - a new source s^* and drain t^*
2. add to G^* each edge $u \to v \in G(E)$
   - with infinite capacity and cost function c
   - and its inverse edge $v \to u$ with the capacity of the original edge $\text{cov}(u \to v)$ and cost function c
3. add to G^* each node $v \in G(V)$
   - compute the difference in coverage between incoming and outgoing edges for v: $d = \sum_{u \to v \in G(E)} \text{cov}(u \to v) - \sum_{v \to u \in G(E)} \text{cov}(v \to u)$
   - if $d > 0$ add the edge $v \to t^*$ at the capacity of d and 0 cost
   - if $d < 0$ add the edge $s^* \to v$ at the capacity of $-d$ and 0 cost

Calculate the maximal flow fl at minimal cost from $s*$ to $t*$. Correct edge coverage in the original graph as $\text{cov}(u \to v) = \text{cov}(u \to v) + \text{fl}(u \to v) - \text{fl}(v \to u)$. While minimal cost flows do in general only allow a linear cost function, quadratic cost functions can be modeled by a further graph transformation. Replace each edge with by a set of b parallel edge $x_1, \ldots, x_b$ each at capacity 1, where b is the capacity of the original edge. Set the cost for each new edge as $\overline{c}(x_i) = c(i)$. The flow over all parallel edges needs to be collected to correct the original coverage.

As a result, we gain a network flow in the splice graph where any flow decomposition into a set of paths is optimal. Parsimoniously, we may attempt to find the decomposition set with the smallest number of paths that

are compatible to he set of bins and pairs. This, however, is NP-hard even using the restricted path enumeration schema described above. Solutions may be approximated at different levels. A minimal flow decomposition may be approximated by either successively removing the longest or the path of maximal flow. We propose the following, possibly more effective, strategies:

- Establish the full set of compatible transcript paths as before. Successively remove either the path denoting the maximal flow or the longest path and remove the respective flow from the graph until no flow is left.
- Dynamic programming may be used to establish the longest path or the path of maximal flow at each step without computing the full path set. As compatible constrains sets from bin and paired reads need to be saved for each node we do not improve on theoretical complexity.
- This procedure is highly efficient, nevertheless, when such constraints are ignored.

It remains unknown at present, whether constraints deriving from multi-splice bins or read pairs can be included into the flow definition. The use of such constraints during the decomposition phase therefore commonly introduces some "leftover flow" assigned to isoforms incompatible with the multi-splice bins. Much like λ and μ regulate the trade-off between sparseness and error in the optimization, the flow decomposition may be aborted if only negligible flow remains to be assigned.

4 The Task

The goal of this course is to create a tool for transcript assembly that allows us to seamlessly interchange components. As a final result, we want to compare different methods and cost functions and thereby explore the landscape of transcripts.

4.1 WP0: Preparation

- You are provided with a set of splice graphs and their respective bins and read pairs in a custom format.
- A Python-based parsing script is also provided. You may modify and extend this script for this project.
- The script includes also functionality to write valid GTF-Files for benchmarking.
- Familiarize yourself with the custom graph format and the GTF-Format required as output.
- The project requires the use of `Python 3`, `Gurobi` with the Python 3 Interface *gurobipy*, and `NetworkX`. Install all tools and familiarize yourself with them.
- A further script allows to compare the GTF file of an assembly to the also provided truth set. We use this for benchmarking the assembly quality. In order to execute this script please install the tool `cuffcompare`.

4.2 WP1: Path Enumeration

- Implement the strategies for path enumeration described above, both with and without constraints from bins and read pairs.
- Include effective "emergency break" mechanisms to avoid excessive runtimes on too complex loci.
- Explore the properties of the computed path-sets. Compute statistics for key properties, including but not limited to:
  - path lengths,
  - set sizes,
  - and common edges between path.
- Consider alternative enumeration schemes, such as starting at a central node, which may be more efficient in combination with bin constraints.

4.3 WP2: Optimization with Linear and Quadratic Programming

- Implement the assembly as an application of Linear or Quadratic Programming on the edges of a splice graph as described above.
- Compare assembly results varying the
  - path enumeration strategies as implemented in WP1,
  - norms,
  - and sparsity constraints.

- Compare results for the complete and pre-filtered splice graphs.

4.4 WP3: Flow-based Optimization

- Implement the assembly as an application of the network flow based graph transformation as described above.
- Extend the path enumeration strategies of WP1 for use in flow decomposition.
- Compare assembly results varying the
  - flow decomposition strategy
  - and the used cost function, ideally relating to previously tested norms.
- Compare results for complete and pre-filtered splice graphs.

4.5 WP4: Interpretation and Further Reasearch

- Analyze and summarize the results of all workpackages.
- Bonus: Extend any workpackage by your own ideas. You are welcome to change the specs of this project after consultation with us if you are interested in a "deeper diver" of a particular task.