Exact Transcript Quantification Over Splice Graphs

Cong Ma[1]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
congm1@andrew.cmu.edu

Hongyu Zheng[1]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
hongyuz1@andrew.cmu.edu

Carl Kingsford[2]
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
carlk@cs.cmu.edu

──── **Abstract** ────

The probability of sequencing a set of RNA-seq reads can be directly modeled using the abundances of splice junctions in splice graphs instead of the abundances of a list of transcripts. We call this model graph quantification, which was first proposed by Bernard et al. (2014). The model can be viewed as a generalization of transcript expression quantification where every full path in the splice graph is a possible transcript. However, the previous graph quantification model assumes the length of single-end reads or paired-end fragments is fixed. We provide an improvement of this model to handle variable-length reads or fragments and incorporate bias correction. We prove that our model is equivalent to running a transcript quantifier with exactly the set of all compatible transcripts. The key to our method is constructing an extension of the splice graph based on Aho-Corasick automata. The proof of equivalence is based on a novel reparameterization of the read generation model of a state-of-art transcript quantification method. This new approach is useful for modeling scenarios where reference transcriptome is incomplete or not available and can be further used in transcriptome assembly or alternative splicing analysis.

──────────

[1] Equal Contribution
[1] Equal Contribution
[2] Corresponding Author

20th International Workshop on Algorithms in Bioinformatics (WABI 2020).
Editors: Carl Kingsford and Nadia Pisanti; Article No. 12; pp. 12:1–12:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1   Introduction

Transcript quantification has been a key component of RNA-seq analysis pipelines, and the most popular approaches (such as RSEM [9], kallisto [4], and Salmon [12]) estimate the abundance of individual transcripts by inference over a generative model from transcripts to observed reads. To generate a read in the model, a transcript is first sampled proportional to its relative abundance multiplied by length, then a fragment is sampled as a subsequence of the transcript according to bias correction models. The quantification algorithm thus takes the reference transcriptome and the set of reads as input and outputs a most probable set of relative abundances under the model. We focus on a generalization of the problem, called graph quantification, that allows for better handling of uncertainty in the reference transcriptome.

The concept of graph quantification was first proposed by Bernard et al. [3], which introduced a method called FlipFlop. Instead of a set of linear transcripts, a splice graph is given and every transcript compatible with the splice graph (a path from transcript start to termination in the splice graph) is assumed to be able to express reads. The goal is to infer the abundance of edges of the splice graph (or its extensions) under flow balance constraints. Transcript abundances are obtained by flow decomposition under this setup. FlipFlop infers network flow on its extension of splice graphs, called fragment graphs, and uses the model to further assemble transcripts. However, the proposed fragment graph model only retains its theoretical guarantee when the lengths of single-end reads or paired-end fragments are fixed. In this work, we propose an alternative approach to graph quantification that correctly addresses the variable-length reads and corrects for sequencing biases. Our method is based on flow inference on a different extension of the splice graph.

Modeling RNA-seq reads directly by network flow on splice graphs (or variants) is advantageous when the set of transcript sequences is uncertain or incomplete. It is unlikely that the set of reference transcripts is correct and complete for all genes in all tissues, and therefore, many transcriptome assembly methods have been developed for reconstructing a set of expressed transcripts from RNA-seq data [18, 13, 10, 14], including FlipFlop [3]. Recent long read sequencing confirms the expression of unannotated transcripts [17], but they also show that the individual exons and splice junctions are relatively accurate. With incomplete reference transcripts but correct splice graphs, it is more appropriate to model RNA-seq reads directly by splice graph network flows compared to modeling using the abundances of an incomplete set of transcripts.

The network flow of graph quantification may be incorporated into other transcriptome assembly methods in addition to FlipFlop. StringTie [13] iteratively finds the heaviest path of a flow network constructed from splice graphs. A theoretical work by Shao et al. [15] studies the minimum path decomposition of splice graphs when the edge abundances satisfy flow balance constraints. Better network flow estimation on splice graphs inspires improvement of transcriptome assembly methods.

The splice graph flow itself is biologically meaningful as it indicates the relative usage of splice junctions. Estimates of these quantities can be used to study alternative splicing patterns under the incomplete reference assumption. PSG [8] pioneered this line of work but with a different abundance representation in splice graph. It models splice junction usage by fixed-order Markov transition probabilities from one exon (or fixed number of predecessor exons) to its successor exon in the splice graph. It develops a statistical model to detect the difference in transition probability between two groups of samples. However, fixed-order Markov chain is less expressive: a small order cannot capture long-range phasing relationships,

and a large order requires inferring a number of transition probabilities that are likely to lack sufficient read support. Markov models set the abundance of a transcript to the product of transition probabilities of its splice junctions, which implicitly places a strong constraint on the resulting transcriptome. Many other previous studies of splice junction usage depend on a list of reference transcripts and compute the widely used metric Percentage Spliced In (PSI) [7, 16, 19]. Under an incomplete reference assumption, the estimated network flow is a potential candidate to compute PSI and study alternative splicing usage.

A key challenge of graph quantification, especially for paired-end reads, is to incorporate the co-existence relationship among exons in transcripts. When a read spans multiple exons, the exons must co-exist in the transcript that generates this read. Such a co-existence relationship is called phasing, and the corresponding read is said to contain phasing information. For these reads, the flows of the spanned splice edges may be different from each other, and in this case, the probability of the read cannot be uniquely inferred from the original splice graph flow. FlipFlop solves this problem by expanding the splice graph into a fragment graph, assuming all reads are fixed-length. In a fragment graph, every vertex represents a phasing path, two vertices are connected if the phasing paths represented by the vertices differ by one exon, and every transcript on the splice graph maps to a path on the fragment graph. The mapped path in the fragment graph contains every possible phasing path from a read in the transcript, in ascending order of genomic location. However, it is not possible to construct this expansion of splice graphs when the reads or fragments are of variable lengths. There is no longer a clear total order over all phasing paths possible from a given transcript, and it is unclear how to order the phasing paths in a fragment graph. We detail the FlipFlop model in Section A.3.

To incorporate the phasing information from variable-length reads or fragments, we develop a dynamic unrolling technique over the splice graph with an Aho-Corasick automaton. The resulting graph is called prefix graph. We prove that optimizing network flow on the prefix graph is equivalent to the state-of-the-art transcript expression quantification formulation when all full paths of splice graphs are provided as reference transcripts, assuming modeled biases of generating a fragment are determined by the fragment sequence itself regardless of which transcript it is from. In other words, quantification on prefix graphs generates exact quantification for the whole set of full splice graph paths. The proof is done by reparameterizing the sequencing read generation model from transcript abundances to edge abundances in the prefix graph. We also propose a specialized EM algorithm to efficiently infer a prefix graph flow that solves the graph quantification problem.

As a case study, we apply our method on paired-end RNA-seq data of bipolar disease sequencing samples and estimate flows for neurogenesis-related genes, which are known to have complex alternative splicing patterns and unannotated isoforms. We use this case study to demonstrate the applicability of our method to handle variable-length fragments. Additionally, the network flow leads to different PSI compared to the one computed with reference transcripts, suggesting reference completeness should be considered in alternative splicing analysis.

2 Methods

We now provide a brief technical overview of the method section.

In Section 2.1, we describe the detailed derivation and procedure to reparameterize the generative model in transcript quantification. A key component in this process is re-defining transcript effective length. The transcript effective length is introduced to offset sampling

biases towards shorter transcripts, and an empirical formula by penalizing transcript length with average fragment length has been widely used. We show that this empirical formula has a more elegant explanation, purely from introspection of the generative model. Based on this observation, we naturally introduce the path abundances, the new set of variables that parameterize the generative model, and the path effective lengths, weights of the path abundances in the normalization constraints. To introduce bias correction, we introduce the concept of affinity that encodes bias corrected likelihood for generating a fragment at a particular location, and the rest follows naturally by redefining the effective lengths.

In Section 2.2, we describe the prefix graph, whose purpose is to map the abundances of compatible transcripts onto network flows that preserve path abundances. This is beneficial, as we avoid enumerating compatible transcripts and only need to infer the prefix graph flow. The key technical contribution in this section is connecting the process of matching phasing paths onto transcripts, to the general problem of multi-pattern matching. This leads to a rollout of the splice graph according to an Aho-Corasick automaton, and the correctness (that the flow preserves of path abundances) can be proved by running the Aho-Corasick algorithm on the compatible transcripts.

In Section 2.3, we describe the inference process for the prefix graph flows, as we need to expand our model to handle multi-mapped reads within a gene or across different genes. We employ a standard EM algorithm for multi-mapped reads, similar to existing approaches. Inference across genes is enabled by another reparameterization of the generative model, which relativizes edge abundances to its incident gene. We are able to decouple the inference for each gene during the M-step, which combined with a simple E-step, allows for efficient inference and completes the specification of our methods.

We formally define the following terms. A **splice graph** is a directed acyclic graph representing alternative splicing events in a gene. The graph has two special vertices: S represents the start of transcripts and T represents the termination of transcripts. Every other vertex represents an exon or a partial exon. Edges in the splice graph represent **splice junctions**, potential adjacency between the exons in transcripts, or connect two adjacent partial exons. A **path** is a list of vertices such that every adjacent pair is connected by an edge, and an $S - T$ path is a path that starts with S and ends with T. We refer to **phasing paths** as the paths of which the exons co-exist in some transcripts. Specifically, we use a generalized notion of phasing path that includes singleton path (path of a single vertex) and path consisting of a single edge, so all vertices and edges are considered phasing paths. Each transcript corresponds to a unique $S - T$ path in the splice graph, and as discussed in the introduction, we will assume every $S - T$ path is also a transcript. Graph quantification generalizes transcript quantification as we can set up a "fully rolled out" splice graph, containing only chains each corresponding to a linear transcript. We use the phrase **quantified transcript set** to denote a set of transcripts with corresponding abundances.

2.1 Reparameterization

Our goal in this section is to establish an alternative set of parameters for the graph quantification problem. In the transcript quantification model, every transcript corresponds to a variable denoting its relative abundance. We will identify a more compact set of parameters that would represent the same model, as described below.

We start with the core model of transcript quantification at the foundation of most modern methods [9, 6, 4, 12]. Assume the paired-end reads from an RNA-seq experiment are error-free and uniquely aligned to a reference genome with possible gaps as fragments (assumptions will be relaxed later). We denote the set of fragments (mapped from paired-

end reads) as F, the set of transcripts as $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ with corresponding lengths $l_1, l_2, \ldots, l_n$ and abundances (copies of molecules) $c_1, c_2, \ldots, c_n$. This can be used to derive other quantities, for example, the transcripts per million (TPM) values are calculated by normalizing $\{c_i\}$ then multiplying the values by 10^6. Under the core model, the probability of observing F is:

$$P(F \mid \mathcal{T}, c) = \prod_{f \in F} \sum_{i \in \text{idx}(f)} P(T_i) P(f \mid T_i).$$

Here, $P(T_i)$ denotes the probability of sampling a fragment from transcript T_i, and $P(f \mid T_i)$ denotes the probability of sampling the fragment f given it comes from T_i. $\text{idx}(f)$ is the set of transcript indices onto which f can map. Let $D(l)$ be the distribution of generated fragment length. In the absence of bias correction, $P(f \mid T_i)$ is proportional to $D(f) = D(l(f))$, where $l(f)$ denotes the fragment length inferred from mapping f to T_i. Define the effective length for T_i as $\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k - j + 1)$ (which can be interpreted as the total "probability" for T_i to generate a fragment), and $P(f \mid T_i) = D(f)/\hat{l}_i$. The probability of generating a fragment from T_i is assumed to be proportional to its abundance times its effective length, meaning $P(T_i) \propto c_i \hat{l}_i$. Our definition of effective length is different from existing literature, where it is usually defined as $l_i - \mu(T_i)$, the actual length of transcript l_i minus the truncated mean of D, and the truncated mean is defined as $\mu(T_i) = (\sum_{j=1}^{l_i} j D(j))/(\sum_{k=1}^{l_i} D(k))$. However, these two definitions are actually essentially the same most of the time:

▶ **Lemma 1.** $\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k - j + 1) = (\sum_{t=1}^{l_i} D(t))(l_i + 1 - \mu(T_i))$.

Proof.

$$\hat{l}_i = \sum_{t=1}^{l_i} D(t)(l_i + 1 - t)$$

$$= (l_i + 1) \sum_{t=1}^{l_i} D(t) - \sum_{t=1}^{l_i} t D(t)$$

$$= \left(\sum_{t=1}^{l_i} D(t) \right) \left(l_i + 1 - \frac{\sum_{t=1}^{l_i} t D(t)}{\sum_{t=1}^{l_i} D(t)} \right)$$

$$= \left(\sum_{t=1}^{l_i} D(t) \right) (l_i + 1 - \mu(T_i))$$

This means ignoring the difference between l_i and $l_i + 1$, the two definitions differ by a multiplicative factor of $\sum_{t=1}^{l_i} D(t)$. ◀

The factor $\sum_{t=1}^{l_i} D(t)$ is the probability of sampling a fragment no longer than l_i. It is very close to 1 as long as the transcript is longer than most fragments, which is usually true in practice. We refer to previous papers [9, 11, 6, 4, 12] for more detailed explanation of the model. This leads to:

$$P(F \mid \mathcal{T}, c) = \prod_{f \in F} (\sum_{i \in \text{idx}(f)} c_i) D(f)/(\sum_{T_i \in \mathcal{T}} c_i \hat{l}_i).$$

We now propose an alternative view of the probabilistic model with paths on splice graphs in order to derive a compact parameter set for the quantification problem. The splice graph is constructed so each transcript can be uniquely mapped to an $S - T$ path $p(T_i)$ on the graph, and we assume the read library satisfies that each fragment f can be uniquely mapped

to a (non $S - T$) path $p(f)$ on the graph (this assumption will also be relaxed later). With this setup, $i \in \mathrm{idx}(f)$ if and only if $p(f)$ is a subpath of $p(T_i)$, or $p(f) \subset p(T_i)$.

We now define $c_p = \sum_{i:T_i \in \mathcal{T}, p \subset p(T_i)} c_i$ to be the total abundance of transcripts including path p, called **path abundance**, and $\hat{l}_p = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} \mathbf{1}(p(T_i[j,k]) = p)D(k - j + 1)$ called **path effective length**, where $T_i[j,k]$ is the fragment generated from transcript i from base j to base k and $\mathbf{1}(\cdot)$ is the indicator function. Intuitively, the path effective length is the total probability of sampling a fragment that maps exactly to the given path. This definition is independent of the chosen transcript T_i and any T_i yields the same result as long as T_i includes p. Next, let $\mathcal{P}$ be the set of paths from the splice graph satisfying $\hat{l}_p > 0$.

▶ **Lemma 2.** *The normalization term can be reparameterized:* $\sum_{T_i \in \mathcal{T}} c_i \hat{l}_i = \sum_{p \in \mathcal{P}} c_p \hat{l}_p$.

Proof. The idea is to break down the expression of $\hat{l}_i$ into a sum over fragments, and regroup the fragments by the path to which they are mapped:

$$
\sum_{T_i \in \mathcal{T}} \hat{l}_i c_i = \sum_{T_i \in \mathcal{T}} \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} D(k - j + 1)c_i
$$
$$
= \sum_{p \in \mathcal{P}} \sum_{i,j,k:p(T_i[j,k])=p} D(k - j + 1)c_i
$$
$$
= \sum_{p \in \mathcal{P}} \Big(\sum_{j,k:\exists i,p(T_i[j,k])=p} D(k - j + 1) \Big) \Big(\sum_{i:p \subset p(T_i)} c_i \Big)
$$
$$
= \sum_{p \in \mathcal{P}} \hat{l}_p c_p
$$

The third equation holds because the sum of $D(k - j + 1)$ across any transcripts containing path p is the same, as a shift in the reference does not change $D(k - j + 1)$ assuming there are no sequencing biases. ◀

The likelihood objective can now be rewritten as:

$$
P(F \mid \mathcal{T}, c) = \prod_{f \in F} \Big(\sum_{j:p(f) \subset p(T_j)} c_j \Big) D(f) / \Big(\sum_{p \in \mathcal{P}} c_p \hat{l}_p \Big)
$$
$$
\propto \prod_{f \in F} c_{p(f)} / \Big(\sum_{p \in \mathcal{P}} c_p \hat{l}_p \Big) \tag{1}
$$

This reparameterizes the model with $\{c_p\}$, the path abundance. In practice, we reduce the size of $\mathcal{P}$ by discarding long paths with small $\hat{l}_p$ and no mapped fragments, as they contribute little to the likelihood (see Section A.2). To incorporate bias correction into our model, we define the affinity $A_p(j, k)$ to be the unnormalized likelihood of generating a read pair mapped to path p from position j to k. This is the analog for $P(f \mid t_i)$ in the transcript quantification model. In the non-bias-corrected model, we simply have $A_p(j, k) = D(k - j + 1)$. Certain motif-based corrections and GC-content-based corrections, which are calculated from the genomic sequence in between the paired-end alignment, can then be integrated into our analysis naturally. To adapt the likelihood model to bias correction, we define transcript and path effective length as follows:

$$
\hat{l}_i = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} A_{p(T_i[j,k])}(j, k)
$$
$$
\hat{l}_p = \sum_{j=1}^{l_i} \sum_{k=j}^{l_i} A_p(j, k) \mathbf{1}(p(T_i[j,k]) = p), \forall p \subset p(T_i)
$$

$\hat{l}_p$ is still the same for any T_i that includes p, so it does not matter which transcript is used to compute it. $p(T_i[j,k])$ denotes the path that $T_i[j,k]$ (transcript T_i from location j to k) maps to, and we assume the coordinate when calculating A_p coincides with that of T_i. The definition of path abundance remains unchanged, and all of our proposed methods will work in the same way. Transcript-specific bias correction requires an approximation to the affinity term, and we discuss this topic in detail in Section A.1.

We have now completed the necessary steps to claim the following theorem, which formally establishes the correctness of the reparameterization procedure with bias correction:

▶ **Theorem 3.** *Assuming each read is uniquely mapped to one phasing path, the following two optimization instances are equivalent:*

- *optimizing $\{c_p\}$, which are the path abundances under the reparameterized objective $\prod_{f\in F} c_{p(f)}/(\sum_{p\in\mathcal{P}} c_p \hat{l}_p)$, conditioned on $\{c_p\}$ corresponding to a valid quantified set of transcripts;*
- *optimizing $\{c_i\}$ which are the transcript abundances under the original objective $\prod_{f\in F}(\sum_{i\in idx(f)} c_i)/(\sum_{T_i\in\mathcal{T}} c_i \hat{l}_i)$.*

Here $\hat{l}_i$ and $\hat{l}_p$ are transcript effective length and path effective length defined with the same set of affinities $A_p(j,k)$.

Proof. This naturally follows in two steps. First, we can prove Lemma 2 with bias correction using the identical technique of breaking $\hat{l}_i$ down to sum over fragments, then regroup by path mappings. This means the normalization term can be reparameterized. We finish by reparameterizing the whole likelihood in the same way as in the non-bias-corrected case (see equation (1)), again with identical technique. ◀

With reads multimapped to different phasing paths (within or across genes), let $M(f)$ denote the set of phasing paths f can map onto, and for $p \in M(f)$ let $A(f \mid p)$ denote the affinity of f mapping to p. In this case, we can use the same idea of grouping transcripts by the phasing path that f maps onto:

$$
\begin{aligned}
P(f) &= \sum_{i\in\mathrm{idx}(f)} P(T_i)P(f \mid T_i) \\
&= \sum_{p\in M(f)} \sum_{i:p\subset T_i} c_i A(f \mid p) \\
&= \sum_{p\in M(f)} c_p A(f \mid p).
\end{aligned}
$$

The reparameterization theorem holds by replacing $c_{p(f)}$ with $\sum_{p\in M(f)} c_p A(f \mid p)$ in the objective function.

2.2 Prefix Graphs

In Theorem 3, we showed that to perform graph quantification, it is sufficient to optimize the path abundances under a reparameterized objective, conditioned on that the path abundances correspond to a quantified set of transcripts. This means to apply the theorem for optimization of path abundance, we need a set of constraints that ensures this condition. One solution is to introduce a variable for every compatible transcript and then use the definition of c_p as the constraints. However, this will lead to an impractically large model, as the number of $S-T$ paths in the splice graph can be exponentially larger than the size of the prefix graph. In this section, we derive a set of linear constraints governing $\{c_p\}$ that achieves this purpose.

To motivate the next step, assume every inferred fragment either resides within an exon or contains one junction. In this case, the phasing paths are nodes or edges in the splice graph. If the quantified transcript set is mapped onto the splice graph, we obtain a network flow. The path abundance for a phasing path equals either the flow through a vertex or an edge. By the flow decomposition theorem, given a network flow on the splice graph, we can decompose it into $S - T$ paths with weights, which then naturally maps back to a quantified transcript set. As the two-way mapping (between quantified transcript sets and splice graph flows) preserves path abundances, we conclude optimization over a splice graph flow would achieve the goal of graph quantification. Specifically, it is easy to restructure the constraints to represent a splice graph flow, and optimizing the resulting model is equivalent to the transcript quantification model with all compatible transcripts included.

This solution no longer works when some phasing path p contains three or more exons. This is because one cannot determine the total flow that goes through two consecutive edges (corresponding to a phasing path with two junctions) just from the flow graph, and different decompositions of the flow lead to different answers. Informally, this can be solved by constructing higher-order splice graphs (as done by Legault et al. [8] for example), or fixed-order Markov models, but the size of the resulting graph grows exponentially fast and some phasing paths can be very long. Instead, we choose to "unroll" the graph just as needed, roughly corresponding to a variable-order Markov model, similar to FlipFlop [3] but applicable to paired-end reads.

To motivate our proposed unrolling method, consider the properties it needs to satisfy. Roughly speaking, the unrolled graph needs to exactly identify every path in $\mathcal{P}$ to accurately calculate the path abundances. That is, for every path p in $\mathcal{P}$, there is a set of vertices or edges in the unrolled graph, such that a transcript includes p if and only if its corresponding $S - T$ path intersects with this set. We can view this "identify phasing paths" problem as an instance of multiple pattern matching. That is, given $\mathcal{P}$, for a given transcript T_i, we want to determine the set of paths in $\mathcal{P}$ that are subpaths of T_i, reading one exon of T_i at a time. Similar to our previous example, if $\mathcal{P}$ contains only single exons, we only need to recognize $[x]$ (the singleton path including only x) when we read exon x, and we will recognize a general phasing path p when the transcript we have seen admits p as a suffix. To speed up the process, we can memorize a suffix of the transcript we have seen that is a prefix of some path in p, so we do not need to check all preceding exons again when trying to recognize p. This is not a new idea and in fact is the Aho-Corasick algorithm [1], a classical algorithm for multiple pattern matching where the nodes in the splice graph (set of exons) is the alphabet, $\mathcal{P}$ is the set of patterns and T_i is the text, and the idea is formalized as a finite state automaton (FSA) that maintains the longest suffix of current text that could extend and match a pattern in the future. This can be regarded as an unrolling of the splice graph, which has the power of exactly matching arbitrarily phasing paths, and a flow on the automaton is the analog of a splice graph flow that also is unrolled enough to recover path abundances, as we will prove in this section.

We formalize the idea. Consider the Aho-Corasick FSA constructed from $\mathcal{P}$, where we further modify the finite state automaton as follows. Transitions between states of the FSA, called dictionary suffix links, indicate the next state of the FSA given the current state and the upcoming character. We do not need the links for all characters (exons), as we know $T_i \in \mathcal{T}$ is an $S - T$ path on the splice graph. If x is the last seen character, the next character y must be a successor of x in the splice graph, and we only generate the state transitions for this set of movements. With an FSA, we now construct a directed graph from its states and transitions as described above:

▶ **Definition 4** (Prefix Graph). *Given splice graph G_S and set of splice graph paths $\mathcal{P}$ (assuming every single-vertex path is in $\mathcal{P}$), we construct the corresponding prefix graph G as follows:*

The vertices V of G are the splice graph paths p such that p is a prefix of some path in $\mathcal{P}$. For $p \in V$, let x be the last exon in p. For every y that is a successor of x in the splice graph, let p' be the longest path in V that is a suffix of py (py is the path generated by appending y to p). We then add an edge from p to p'.
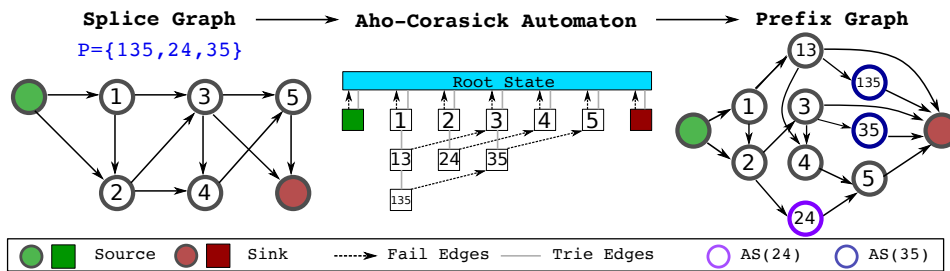
The source and sink of G are the vertices corresponding to splice graph paths $[S]$ and $[T]$, where $[x]$ denotes a single-vertex path. The set $AS(p)$ is the set of vertices p' such that p is a suffix of p'.

Intuitively, the states of the automaton are the vertices of the graph and are labeled with the suffix in consideration at that state. The edges of the graph are the dictionary suffix links of the FSA, now connecting vertices. For $p \in \mathcal{P}$, $AS(p)$ denotes the set of states in FSA that recognizes p. All transcripts start with S, end with T and there is no path in $\mathcal{P}$ containing either of them as they are not real exons, so there exist two vertices labeled $[S]$ and $[T]$. We call them the source and sink of the prefix graph respectively, and we will see they indeed serve a similar purpose.

▶ **Lemma 5.** *There is a one-to-one correspondence between $S - T$ paths in the splice graph and $[S] - [T]$ paths in the prefix graph.*

Proof. Every transcript can be mapped to an $[S] - [T]$ path on the prefix graph by feeding the transcript to the finite state automaton and recording the set of visited states, excluding the initial state where no string is matched. The first state after the initial state is always $[S]$ as the first vertex in an $S - T$ path is S, and the last state is always $[T]$ because there are no other vertexes in the prefix graph that would contain T. Conversely, a $[S] - [T]$ path on the prefix graph can also be mapped back to a transcript, as it has to follow dictionary suffix links (transitions between FSA states), which by our construction can be mapped back to edges in the splice graph. ◀

This implies that the prefix graph is also a DAG: If there is a cycle in the prefix graph, it implies an exon appears twice in a transcript, which violates our assumption that the splice graph is a DAG.

Figure 1 An example construction of the Prefix Graph. The source and sink of the prefix graph are $[S]$ and $[T]$, respectively. The set of phasing paths $\mathcal{P}$ is shown in blue in the left panel, and we does not include the singleton paths for simplicity. We draw the trie and the fail edges for the A-C automaton as it reduces cluttering (dictionary suffix link can be derived from both edge sets). The colored nodes in prefix graph are the vertices (states) in $AS(35)$ and $AS(24)$.

The resulting prefix graph flow serves as a bridge between the path abundance $\{c_p\}$ and the quantified transcript set $\{c_i\}$:

▶ **Theorem 6.** *Every quantified transcript set can be mapped to and from a prefix graph flow.
The path abundance is preserved during the mapping and can be calculated exactly from prefix
graph flow: $c_p = \sum_{s \in AS(p)} f_s$, where f_s is the flow through vertex s.*

Proof. Using the path mapping between splice graph and prefix graph, we can map a
quantified transcript set onto the prefix graph as a prefix graph flow and reconstruct a
quantified transcript set by decomposing the flow and map each $[S] - [T]$ path back to the
splice graph as a transcript.

To prove the second part, let $\{c_p\}$ be the path abundance calculated from the definition
given a quantified transcript set, and $\{c'_p\}$ be the path abundance calculated from the prefix
graph flow. We will show $\{c_p\} = \{c'_p\}$ for any finite decomposition of the prefix graph flow.

For any transcript T_i and any path $p \in \mathcal{P}$, since no exon appears twice for a transcript, if
T_i contains p, it will be recognized by the FSA exactly once. This means the $[S] - [T]$ path to
which T_i maps intersects with $AS(p)$ by exactly one vertex in this scenario, and it contributes
the same abundance to c'_p and c_p. If T_i does not contain p, by similar reasoning, it contributes
to neither c'_p nor c_p. This holds for any transcript and any path, so the two definitions of
path abundance coincide and are preserved in mapping from quantified transcript set to
prefix graph flow. Since the prefix graph flow is preserved in flow decomposition, the path
abundance is preserved as a function of prefix graph flow.   ◀

This connection allows us to directly optimize over $\{c_p\}$ by using the prefix graph flow
as variables (the path abundances c_p is now represented as seen in Theorem 6), and use
flow balance and non-negativity as constraints, as we describe in the next section. The
corresponding quantified transcript set is guaranteed to exist by a flow decomposition followed
by the mapping process.

We next describe an improvement to the prefix graph, which we call compact prefix graph.
The idea is to recognize phasing paths at the edges of the resulting graph, instead of at the
vertices. We will still start with the Aho-Corasick FSA, but we will be building the graph in
a way that states of the FSA correspond to edges of the resulting graph, as described below:

▶ **Definition 7** (Compact Prefix Graph). *Given splice graph G_S and set of splice graph paths
$\mathcal{P}$, we construct the corresponding compact prefix graph G' as follows. The vertex set of the
compact prefix graph is the union of*

- *all single-vertex paths on the splice graph;*
- *any splice graph path p that is the prefix of some path p' in $\mathcal{P}$, while strictly shorter than
  p'.*

*For p in the compact prefix graph, let x be its last exon and y be a successor of x in the splice
graph. We create an edge which has label py (again, appending y to p), originates from p,
and leads to the node that is the longest suffix of py in the compact prefix graph.*

*The source and sink of G are the vertices corresponding to splice graph paths $[S]$ and $[T]$.
The set $AS(p)$ is the set of **edges** p' such that the edge label on p' is a suffix of p.*

The set $AS(p)$ bears the same meaning as in the original prefix graph, as the states of
the Aho-Corasick FSA are now (roughly) the edges of the compact prefix graph. With this
intuition, we can prove the same property as stated in Lemma 5 and Theorem 6 for compact
prefix graph. The compact prefix graph by the virtue of its construction is a smaller graph
(compared to the original prefix graph) with the same power and is preferred in practice.

2.3   Inference

While the restructuring process described in the previous section reduces the size of the optimization problem, we still need to solve it efficiently. We start with the base case, that is, a single gene and every read pair maps to exactly one path. Recall that $\mathcal{P}$ is the set of phasing paths we consider, $p(f)$ is the path fragment f maps to. For a phasing path p, c_p is the path abundance, $\hat{l}_p$ is the path effective length. For the prefix graph, we let f_e denote the flow through an edge, f_v denote the flow through a vertex, and $AS(p)$ is the set of vertices (as FSA states) that recognize p. We also use $\text{In}(v)$ to denote the incoming edges of vertex v, and $\text{Out}(v)$ similarly for the outgoing edges. The full instance in this case, with the prefix graph proposed the previous section, is:

$$
\begin{aligned}
\max \quad & \sum_{f \in F} \log c_{p(f)} \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}} c_p \hat{l}_p = 1 \\
& c_p = \sum_{v \in AS(p)} f_v && \forall p \in \mathcal{P} \\
& f_v = \sum_{e \in \text{In}(v)} f_e = \sum_{e \in \text{Out}(v)} f_e && \forall v \in V - \{[S], [T]\} \\
& f_e \geq 0 && \forall e \in E
\end{aligned}
$$

This is slightly different from what we described in Section 2.1. First, we maximize the logarithm of the likelihood objective. Second, we explicitly fix the normalization constant to be 1, instead of placing it on the divisor of the fragment likelihood. This does not change the objective, and the only difference is that in the original form $\{c_p\}$ can be arbitrarily scaled, while here the scaling is fixed. The variables and the constraints come from the prefix graph flow, and c_p is represented as in Theorem 6. For a compact prefix graph as described in Definition 7, we simply replace the equation of c_p to sum over f_e with $e \in AS(p)$. This is a convex problem, as the target function is convex with respect to $\{c_p\}$, and the constraints are all linear. We can solve the problem with general purpose convex solvers.

With the presence of multimapped reads (to multiple genes and/or multiple paths within one gene), we can employ a standard EM approach. Recall $M(f)$ is the set of phasing paths onto which f can map, and for $p \in M(f)$ let $A(f \mid p)$ denote the affinity of f mapping to p, as described in Section 2.1. We also let z denote the hidden allocation vector, where $z_{f,p}$ denotes the probability that fragment f is mapped onto splice graph path p. We can alternatively optimize for $\{z_{f,p}\}$ and $\{c_p\}$ until convergence as follows:

$$
\begin{aligned}
z_{f,p}^{(t)} &= c_p^{(t)} A(f \mid p) / \Big(\sum_{p' \in M(f)} c_{p'}^{(t)} A(f \mid p') \Big) \\
c^{(t+1)} &= \arg\max_c \sum_{p \in \mathcal{P}} \Big(\sum_{f \in F} z_{f,p}^{(t)} \Big) \log c_p, \text{s.t.} \sum_{p \in \mathcal{P}} c_p \hat{l}_p = 1
\end{aligned}
$$

$z^{(t)}$ and $c^{(t)}$ denote the variables at iteration t. We hide the constraint from prefix graphs for clarity. The optimization for $z_{f,p}^{(t)}$ can be run in parallel, so we focus on the M-step that optimizes $c^{(t+1)} = \{c_p^{(t+1)}\}$, hiding the superscript whenever it is clear from context. When we optimize over the whole genome, the instance becomes impractically huge. This is because we need to infer the flow for every prefix graph (one for each gene) across the whole genome, and we need to satisfy flow balance for each graph and normalization for all graphs together.

We let $\mathcal{G}$ denote the set of genes. Denote the gene abundance $c_g = \sum_{p \in \mathcal{P}_g} c_p \hat{l}_p$, where $\mathcal{P}_g$ is the set of phasing paths in gene g. We then define relative abundance $c_p^* = c_p / c_g$ for every phasing path p. Plugging $c_p = c_g c_p^*$ into the expression for M-step, we have the following transformed objective:

$$
\max \quad \sum_{p \in \mathcal{P}} (\sum_{f \in F} z_{f,p}^{(t)})(\log c_p^* + \log c_g)
$$

$$
= \sum_{g \in \mathcal{G}} \sum_{p \in \mathcal{P}_g} (\sum_{f \in F} z_{f,p}^{(t)}) \log c_p^* + \sum_{g \in \mathcal{G}} (\sum_{f \in F} \sum_{p \in \mathcal{P}_g} z_{f,p}^{(t)}) \log c_g
$$

$$
= \sum_{g \in \mathcal{G}} \sum_{p \in \mathcal{P}_g} (\sum_{f \in F} z_{f,p}^{(t)}) \log c_p^* + \sum_{g \in \mathcal{G}} s_g \log c_g
$$

$$
\text{s.t.} \quad \sum_{p \in \mathcal{P}} c_g c_p^* \hat{l}_p = \sum_{g \in \mathcal{G}} c_g = 1
$$

$$
\sum_{p \in \mathcal{P}_g} c_p^* \hat{l}_p = 1, \forall g \in \mathcal{G}
$$

Here, $s_g = \sum_{f \in F} \sum_{p \in \mathcal{P}_g} z_{f,p}^{(t)}$ can be interpreted as the estimated read count of gene g. Again for clarity we hide the prefix graph constraints for c_p^*, which retain their original form because all prefix graph constraints are affine. Now, we can decouple optimization of c_p^* and c_g, as the objective function is split into two parts, and each constraint only involves one of them. The optimization for c_p^* can be done for each gene independently, and it is exactly the single-gene optimization as we described above except we weight $\log c_{p(f)}$ with $z_{f,p}^{(t)}$ in the objective. The optimization for c_g has the form $\max \sum_{g \in \mathcal{G}} s_g \log c_g$ constrained by $\sum_{g \in \mathcal{G}} c_g = 1$, from which we derive that $c_g \propto s_g$. Since $\sum_{g \in \mathcal{G}} s_g = \sum_{p \in \mathcal{P}} \sum_{f \in F} z_{f,p}^{(t)} = \sum_{f \in F} 1 = |F|$, we have the following localized EM algorithm:

$$
\text{Global E-step:} \quad z_{f,p}^{(t)} = c_p^{(t)} A(f \mid p) / (\sum_{p' \in M(f)} c_{p'}^{(t)} A(f \mid p'))
$$

$$
\text{Gene-Level M-step:} \quad c^{(t+1)} = \arg\max_c \sum_{p \in \mathcal{P}_g} (\sum_{f \in F} z_{f,p}^{(t)}) \log c_p
$$

$$
\text{s.t.} \sum_{p \in \mathcal{P}_g} c_p \hat{l}_p = \sum_{p \in \mathcal{P}_g} \sum_{f \in F} z_{f,p}^{(t)} / |F|, \forall g \in \mathcal{G}
$$

The M-step is run independently for each gene and can be parallelized. Again we omit listing the prefix graph constraint over c_p for clarity, and c_g is implicitly derived as the right-hand side of the normalization constraint.

3 Experiments

Based on the expression quantification method Salmon [12] and its effective lengths, we implement our method and call it Graph Salmon. We apply Graph Salmon on three bipolar disease (BD) RNA-seq samples and three control samples to estimate the expression network flow on neurogenesis-related genes (GO:0022008), which are known to have complex alternative splicing patterns and novel isoforms. We use this as a case study to show that Graph Salmon is applicable with variable fragment lengths and that the relative usage of splice junctions under the incomplete reference assumption are different from those under complete reference assumption.

3.1 Implementation

The splice graphs are constructed using the reference exons and splice junctions of Gencode [5] version 26. Since Salmon's effective lengths are needed for path effective lengths, we first run Salmon on the samples. We also use Salmon read mappings (obtained with the `-writeMappings` argument) and convert their coordinates onto splice graph nodes and edges. Prefix graphs are constructed with the converted read mappings. Each edge in the prefix graph corresponds to a path in the original splice graph, and we compute the path effective length by taking the average of the effective lengths of the corresponding region in reference transcripts that include the corresponding path (for details see Supplementary Material Section A.1). With the converted read mappings and path effective lengths, the probabilistic model of graph quantification can be specified.

Since only neurogenesis-related genes are of interest and the rest of the genes are assumed to have complete reference transcripts, we assume that Salmon correctly estimates the probability of each paired-end read generated from each gene when the read is mapped to multiple genes. We use Salmon's gene-level weight assignment as read count and only solve the flow optimization problem within each gene, which corresponds to one round of the gene-level M step for each gene.
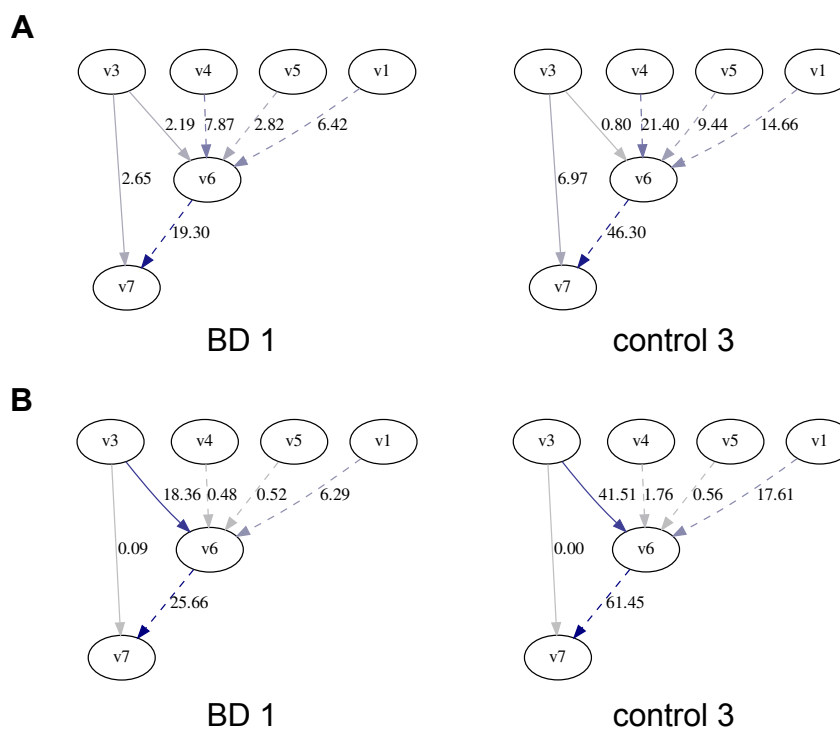
3.2 Graph Salmon reveals unique between-sample differences of PSI for neurogenesis genes

The RNA-seq data can be accessed from Gene Expression Omnibus (GEO) database with accession numbers GSM1288369, GSM1288370, GSM1288371 for bipolar disease samples, and GSM1288374, GSM1288375, GSM1288376 for control samples [2].

The mean fragment lengths of the six sample range from 349.17 bp to 375.28 bp. The standard deviations of fragment lengths are between 53.00 bp and 82.15 bp. Meanwhile, 30% of the exons (or subexons) across the splice graphs are less than 56 bp long, and the 40% quantile of subexon lengths is 79 bp. Graph Salmon is needed in this dataset because of the large standard deviation of fragment lengths compared to subexon lengths.

We computed Percentage Spliced In (PSI) of 2441 skipped exon events using the Graph Salmon network flow and compare them with PSIs calculated using Salmon's expression quantification based on the reference transcripts. Given three exons, the PSI is defined as the total abundance of transcripts that include all three of them, divided by total abundance of transcripts that include the first and the last (but not necessarily the middle one). The correlations of Graph Salmon PSI and Salmon PSI of the same sample are around 0.51 to 0.57 (for both Spearman and Pearson), while the correlations of PSI between different samples computed by the same quantification method are over 0.75 (for both Spearman and Pearson and both methods). The large correlation between different samples can be explained by the fact that they are from the same tissue and should follow the tissue-specific expression and alternative splicing patterns. The smaller correlation between different quantification methods indicates the incomplete reference and complete reference assumptions lead to very different splice junction abundance estimates.

An example of different PSI computed by Graph Salmon and Salmon is shown in Figure 2 and Supplementary Figure A2 on *LPAR1* gene. *LPAR1* gene encodes a lysophosphatidic acid (LPA) receptor that functions in the LPA signaling pathway, which is related to cognitive behavioral deficits such as schizophrenia and depression when dysregulated [20]. We focus on the event that describes the percentage of expression of the inclusions of exon 6 (position 110973480-110973558 in GRCh38) between exon 3 (position 111037840-111038043 in GRCh38)

Figure 2 (A) Network flow of BD 1 and control 3 samples estimated by Graph Salmon. The subgraph includes exons 1, 3 to 7, and exons are represented by nodes and node label indicates the index of exon. PSI of inclusion of exon 6 between exon 3 and 7 is computed. Edges of which the flows are involved in PSI calculation are solid; the rest edges are dashed. (B) Network flow of the same samples computed by Salmon with reference transcripts.

and exon 7 (position 110972072-110972220 in GRCh38). Graph Salmon computes the PSIs to be 0.45 to 0.64 for BD samples and 0.07 to 0.33 for control samples, whereas PSIs computed by Salmon are larger than 0.95 for all six samples.

Even though this difference is not evaluated by rigorous statistical testing, it indicates that when reference is incomplete, previous reference-based alternative splicing analysis may lead to different results. Considering the incomplete reference assumption in alternative splicing analysis enlarges the pool of candidate alternative splicing events.

4   Discussion

We improve the graph quantification model of FlipFlop to incorporate phasing information from variable length reads or fragments. The key algorithmic contributions are a provably correct reparameterization process and the introduction of the prefix graph inspired by Aho-Corasick automata for inference.

To demonstrate the feasibility of our method to handle variable length fragments, we apply our method to neurogenesis-related genes of bipolar disease RNA-seq samples and control RNA-seq samples. The RNA-seq samples contain paired-end reads with mean fragment lengths around 350 bp and standard deviation around 53 – 82 bp. We show that our method successfully estimates network flows on prefix graphs and the estimated flow (under the incomplete reference assumption) only has around 0.5 correlation (both Pearson and Spearman) with the flow estimated by Salmon under the complete reference assumption.

The size of the prefix graph depends on the length of the phasing paths exponentially. Unfortunately, for long read sequencing, especially with transcript-long reads, the prefix graph may be as large as the set of all $S - T$ paths (equivalently the set of all possible transcripts) and its efficiency compared to the naïve implementation of graph quantification (where we enumerate every compatible transcript) may diminish. It is still open what algorithmic tools are required to avoid this ineffiency.

An intrinsic issue with graph quantification is non-identifiability: Many configurations of transcript abundances lead to the same read generation model, and thus it is impossible to distinguish which configuration is closer to the ground truth if our goal is to recover an underlying transcriptome. While our prefix graph representation is compact, for many downstream analyses, we are invariably forced to perform a flow decomposition to transform prefix graph flow into quantified transcript sets. The non-identifiability problem manifests in this step, as different decompositions can lead to the same prefix graph flow, which as we proved implies the same model of read generation. Therefore, it is possible to assess the severity of non-identifiability problem by inspecting different ways of decomposing a fixed prefix graph flow.

This work focuses on theoretical improvements of the graph quantification model, while its practical utility is still largely unexplored. For example, our proposed approach may be a promising method for transcript assembly similar to FlipFlop, where we use quantification for assembly. The method also has potential use cases in alternative splicing analyses and other related tasks in RNA-seq. However, careful benchmarking is needed to determine the cases when graph quantification is superior to standard quantification with a given set of transcripts.

References

1 Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.
2 N Akula, J Barb, X Jiang, JR Wendland, KH Choi, SK Sen, L Hou, DTW Chen, G Laje, K Johnson, et al. RNA-sequencing of the brain transcriptome implicates dysregulation of neuroplasticity, circadian rhythms and GTPase binding in bipolar disorder. *Molecular Psychiatry*, 19(11):1179–1185, 2014.
3 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014.
4 Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016.
5 Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. GENCODE reference annotation for the human and mouse genomes. *Nucleic Acids Research*, 47(D1):D766–D773, 2018.
6 James Hensman, Panagiotis Papastamoulis, Peter Glaus, Antti Honkela, and Magnus Rattray. Fast and accurate approximate inference of transcript expression from RNA-seq data. *Bioinformatics*, 31(24):3881–3889, 2015.
7 Yarden Katz, Eric T Wang, Edoardo M Airoldi, and Christopher B Burge. Analysis and design of RNA sequencing experiments for identifying isoform regulation. *Nature Methods*, 7(12):1009, 2010.
8 Laura H LeGault and Colin N Dewey. Inference of alternative splicing from RNA-Seq data with probabilistic splice graphs. *Bioinformatics*, 29(18):2300–2310, 2013.
9 Bo Li and Colin N Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 2011.

10   Juntao Liu, Ting Yu, Tao Jiang, and Guojun Li. TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs. *Genome Biology*, 17(1):213, 2016.

11   Lior Pachter. Models for transcript quantification from RNA-Seq. *arXiv preprint arXiv:1104.3889*, 2011.

12   Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017.

13   Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

14   Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35(12):1167–1169, 2017.

15   Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2019.

16   Shihao Shen, Juw Won Park, Zhi-xiang Lu, Lan Lin, Michael D Henry, Ying Nian Wu, Qing Zhou, and Yi Xing. rMATS: robust and flexible detection of differential alternative splicing from replicate RNA-Seq data. *Proceedings of the National Academy of Sciences*, 111(51):E5593–E5601, 2014.

17   Manuel Tardaguila, Lorena De La Fuente, Cristina Marti, Cécile Pereira, Francisco Jose Pardo-Palacios, Hector Del Risco, Marc Ferrell, Maravillas Mellado, Marissa Macchietto, Kenneth Verheggen, et al. SQANTI: extensive characterization of long-read transcript sequences for quality control in full-length transcriptome identification and quantification. *Genome Research*, 28(3):396–411, 2018.

18   Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

19   Juan L Trincado, Juan C Entizne, Gerald Hysenaj, Babita Singh, Miha Skalic, David J Elliott, and Eduardo Eyras. SUPPA2: fast, accurate, and uncertainty-aware differential splicing analysis across multiple conditions. *Genome Biology*, 19(1):40, 2018.

20   Yun C Yung, Nicole C Stoddard, Hope Mirendil, and Jerold Chun. Lysophosphatidic acid signaling in the nervous system. *Neuron*, 85(4):669–682, 2015.

A   Additional Methods

In Section A.1 and Section A.2, we describe two modifications to our proposed model in practice. These modifications greatly improve practicality of our proposed model. On the other hand, inclusion of these modifications means the conditions for Theorem 3 no longer hold and the inference is no longer over the exact set of all possible transcripts with exact bias correction. In Section A.3, we describe the fragment graph constructed in FlipFlop in more detail, and describe our reasoning why it loses the theoretical guarantee in presence of variable-length reads or fragments.

A.1   Bias Correction, Continued

Admittedly, there is no way to know the exact location of the read pair within the transcript after reparameterization with path abundance, so these specific biases cannot be integrated into our proposed models directly. Nonetheless, since the splice graph is known in full, approximate bias correction is possible. Let $B_i(j, k)$ denote the affinity value calculated from a full bias correction model for the fragment generated from

base j to k on transcript T_i, and $\hat{r}_i$ be the reference abundance of transcript T_i. We let $A_p(j,k) = (\sum_{i:p \subset p(T_i)} \hat{r}_i B_i(j',k'))/(\sum_{i:p \subset p(T_i)} \hat{r}_i)$, where j' and k' are the coordinates of the path sequence in the reference coordinates of T_i. This means we average the affinity value calculated from known transcripts on this locus, weighted by their reference abundance. For simplicity, we use Salmon outputs as reference abundance, but other approaches, including an iterative process of estimating $A_p(j,k)$ and c_p alternatively, are possible.

One limiting factor for bias correction in our proposed framework is that calculating $\hat{l}_p$ can be expensive as we need to calculate the affinity value for every possible fragment. While we did this for our experiments, there are also alternative approaches that speeds up the process by approximating $\hat{l}_p$. We can use a simplified form for $A_p(j,k)$ so $\hat{l}_p$ has a closed-form solution (for example, do not allow bias correction when calculating effective length, similar to existing approaches to calculate effective length of transcripts), sample from possible $A_p(j,k)$ when the number of fragments from a particular path is large, or precompute the values for fixed genome and bias correction model. These approaches may result in slightly inaccurate path effective length, and it is still an open question how it affects downstream procedures.

A.2 Trimming Set of Phasing Paths

Recall the likelihood function under our reparameterized model:
$P(F \mid \mathcal{T}, c) \propto \prod_{f \in F} c_{p(f)}/(\sum_{p \in \mathcal{P}} c_p \hat{l}_p)$. Paths $p \in \mathcal{P}$ with no mapped fragments do not contribute to $\prod_{f \in F} c_{p(f)}$, as there are no $f \in F$ such that $p(f) = p$. These paths do play a role in calculating the normalization constant $\sum_{p \in \mathcal{P}} c_p \hat{l}_p$. However, since we only remove paths with very low $\hat{l}_p$, the contribution of $c_p \hat{l}_p$ from this set is small. This removal thus causes small underestimation of the normalization constant, and in turn small overestimation of transcript abundances (and path abundances).

If there is a removed path with large c_p when optimized under this model, it means in the inferred quantified transcript set there is are many fragments mapped to path p, even though exactly zero fragments are mapped to p in the sequencing library. This mostly happens if there is a dominant transcript with high abundance, and p is part of the transcript. For such things to happen, the transcript must have many fragments mappable, and the fact that no single fragment mapped to path p indicates $\hat{l}_p$ is small, or the modeling might be faulty. This trimming is necessary in practice, as the fragment length distribution $D(l)$ usually has a long tail when inferred from experiments, due to smoothing and potential mapping errors, leading to many extremely long paths that are near impossible to sample a read from.

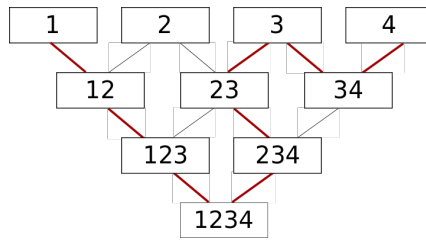## A.3 FlipFlop and the Fragment Graph

The fragment graph constructed by FlipFlop is defined as follows. Given splice graph G and a set of phasing paths $\mathcal{P}$ (again we consider a general notion of phasing paths, meaning single exon paths also count as phasing), the fragment graph G_F is constructed such that

- Each vertex in G_F is either S, T, or a phasing path.
- There is an edge connecting X to Y only if Y is a single exon extension or shrinking of X (unless one of them is either S or T).
- Every $S - T$ path in the splice graph can be mapped uniquely to a $S - T$ path in the fragment graph, and the set of vertices included in the $S - T$ path is exactly the set of phasing paths that are a subpath of the transcript.

We only list a necessary condition in the second item, and we will not discuss how the graph is constructed in practice here. The third item is essential for the FlipFlop algorithm, as it solves the inference problem with convex cost flows which requires that every phasing path is represented by a single vertex in the graph.
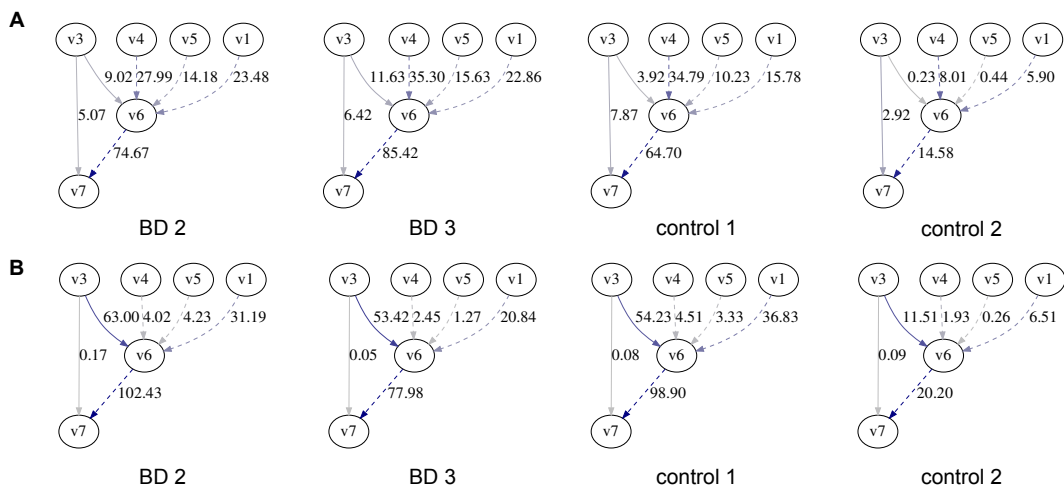
As discussed in the introduction, the FlipFlop algorithm is correct when input library is single-end reads with fixed read length. This implies that if X is a phasing path, there are no phasing paths that are extension of X on both ends (for example, if $X = [3, 4]$, then $[2, 3, 4, 5]$ cannot be a phasing path), otherwise it would violate the condition that all reads have equal length. We now show that there exists no correct fragment graph when the condition is violated.

■ **Figure A1** The fragment graph with 4 exons and 10 phasing paths, not including S and T. Blocks denote vertices of the fragment graph, and lines denote possible edges between vertices (phasing paths). A path visiting 9 vertices (excluding the singleton phasing path [2]) is marked in dark red, and there is no Hamiltonian path in the graph.

Consider a splice graph with a chain of four exons denoted $1, 2, 3$ and 4, and where every subpath of $[1, 2, 3, 4]$ is a phasing path. The fragment graph, if exists, will contain 10 vertices (excluding S and T) and a Hamiltonian path corresponding to the transcript $[1, 2, 3, 4]$. However, as seen in the above figure, the graph will not contain a Hamiltonian path no matter how the graph is constructed.

B   Additional Figure

■ **Figure A2** (A) Network flow of BD 2, BD 3, control 1, and control 2 samples estimated by Graph Salmon. The subgraph includes exons 1, 3 to 7, and exons are represented by nodes and node label indicates the index of exon. PSI of inclusion of exon 6 between exon 3 and 7 is computed. Edges of which the flows are involved in PSI calculation are solid; the rest edges are dashed. (B) Network flow of the same samples computed by Salmon with reference transcripts.