Department of Computer Science
Series of Publications A
Report A-2017-6

Third-generation RNA-sequencing analysis: graph alignment and transcript assembly with long reads

Anna Kuosmanen

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XV, University Main Building, on December 20th, 2017, at 12 o'clock noon.

University of Helsinki
Finland

Supervisors

Veli Mäkinen, University of Helsinki, Finland
Alexandru I. Tomescu, University of Helsinki, Finland

Pre-examiners

Liliana Florea, Johns Hopkins University, USA
Hélène Touzet, University of Lille 1, France

Opponent

Paola Bonizzoni, Università de Milano-Bicocca, Italy

Custos

Veli Mäkinen, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911, telefax: +358 9 876 4314

ISSN 1238-8645
ISBN 978-951-51-3888-0 (paperback)
ISBN 978-951-51-3889-7 (PDF)
Computing Reviews (1998) Classification: G.2.2, I.6.5, J.3
Helsinki 2017
Unigrafia

Third-generation RNA-sequencing analysis: graph alignment and transcript assembly with long reads

Anna Kuosmanen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
anna.kuosmanen@cs.helsinki.fi
http://cs.helsinki.fi/u/aekuosma/

PhD Thesis, Series of Publications A, Report A-2017-6
Helsinki, December 2017, 64+69 pages
ISSN 1238-8645
ISBN 978-951-51-3888-0 (paperback)
ISBN 978-951-51-3889-7 (PDF)

Abstract

The information contained in the genome of an organism, its DNA, is expressed through transcription of its genes to RNA, in quantities determined by many internal and external factors. As such, studying the gene expression can give valuable information for e.g. clinical diagnostics.

A common analysis workflow of RNA-sequencing (RNA-seq) data consists of mapping the sequencing reads to a reference genome, followed by the transcript assembly and quantification based on these alignments. The advent of second-generation sequencing revolutionized the field by reducing the sequencing costs by 50,000-fold. Now another revolution is imminent with the third-generation sequencing platforms producing an order of magnitude higher read lengths. However, higher error rate, higher cost and lower throughput compared to the second-generation sequencing bring their own challenges. To compensate for the low throughput and high cost, hybrid approaches using both short second-generation and long third-generation reads have gathered recent interest.

The first part of this thesis focuses on the analysis of short-read RNA-seq data. As short-read mapping is an already well-researched field, we focus on giving a literature review of the topic. For transcript assembly we propose a novel (at the time of the publication) approach of using minimum-cost flows to solve the problem of covering a graph created from the read alignments

with a set of paths with the minimum cost, under some cost model. Various network-flow-based solutions were proposed in parallel to, as well as after, ours.

The second part, where the main contributions of this thesis lie, focuses on the analysis of long-read RNA-seq data. The driving point of our research has been the Minimum Path Cover with Subpath Constraints (MPC-SC) model, where transcript assembly is modeled as a minimum path cover problem, with the addition that each of the chains of exons (subpath constraints) created from the long reads must be completely contained in a solution path. In addition to implementing this concept, we experimentally studied different approaches on how to find the exon chains in practice. The evaluated approaches included aligning the long reads to a graph created from short read alignments instead of the reference genome, which led to our final contribution: extending a co-linear chaining algorithm from between two sequences to between a sequence and a directed acyclic graph.

Computing Reviews (1998) Categories and Subject Descriptors:
G.2.2 Graph theory: Graph algorithms
I.6.5 Model development
J.3 Life and medical sciences: Biology and genetics

General Terms:
algorithms, design, bioinformatics, computational biology

Additional Key Words and Phrases:
network flow, dynamic programming, directed acyclic graphs

Acknowledgements

These four years have included very many great things and some difficult things. I could not have made it this far without the support of many people, to whom I am deeply thankful.

First, and foremost, I want to thank my supervisors Veli Mäkinen and Alexandru I. Tomescu for all the guidance and support, as well as the freedom to work when and where was best for me. And thanks to my mentor Antti Honkela for encouraging me on this path towards becoming an independent researcher.

I would also like to express my gratitude to all the members, past and present, of the Genome-scale Algorithmics research group; it has been great working with you all.

This dissertation is based on a series of articles, and I am thankful to all of my co-authors. I was not the most reliable person to work with, with all the unpredictable health issues, and you were always there to keep the projects going when I could not.

I would also like to thank my pre-examiners Liliana Florea and Hélène Touzet for their helpful feedback, and Marina Kurtén for suggestions on how to improve the quality of English of this dissertation.

I wish to thank the institutions that have supported this research: the Doctoral Programme in Computer Science (DoCS), the Department of Computer Science of the University of Helsinki, and the Helsinki Institute for Information and Technology (HIIT). Especially I would like to thank Pirjo Moen, the coordinator of DoCS, for all the help in the practical matters related to doctoral studies.

The financial support of the Academy of Finland (Centre of Excellence in Cancer Genetics) and Google is gratefully acknowledged.

And last, but not the least, I want to thank my parents and brothers for their loving support that kept me going.

Helsinki, November 2017
Anna Kuosmanen

Original Papers

The thesis is based on the following five peer-reviewed publications, referred to as Paper I-V. These publications are reproduced at the end of the thesis. None of the publications have been used in previous dissertations.

I. Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. **A Novel Min-Cost Flow Method for Estimating Transcript Expression with RNA-Seq**. *BMC Bioinformatics*, 14(Suppl 5):S15 (10 April 2013). Presented at *RECOMB-Seq* 2013.

II. Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi and Veli äkinen. **A Novel Combinatorial Method for Estimating Transcript Expression with RNA-Seq: Bounding the Number of Paths**. In *Proc. WABI (Workshop on Algorithms in Bioinformatics) 2013*, Springer, LNCS 8126, pp. 85-98, 2013.

III. Anna Kuosmanen, Ahmed Sobih, Romeo Rizzi, Veli Mäkinen, and Alexandru I. Tomescu **On Using Longer RNA-seq Reads to Improve Transcript Prediction Accuracy**. In *Proc. of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016)*. Vol. 3: BIOINFORMATICS, 272-277, 2016

IV. Anna Kuosmanen, Tuukka Norri, and Veli Mäkinen. **Evaluating approaches to find exon chains based on long reads**. *Briefings in Bioinformatics*, 2017, bbw137. doi: 10.1093/bib/bbw137

V. Anna Kuosmanen, Topi Paavilainen, Travis Gagie, Rayan Chikhi, Alexandru I. Tomescu, and Veli Mäkinen. **From the Minimum Path Cover Problem to Boosting Dynamic Programming on DAGs**. Submitted.

Contents

Chapter 1

Introduction

Gene expression controls the functions of all the cells in an organism. The information contained in the genome of an organism, its DNA, is expressed through *transcription* of its genes, in quantities determined by many internal and external factors. This collection of expressed transcripts, the *transcriptome*, can be studied with many different techniques, the earliest ones being hybridization-based *microarrays* and *Sanger sequencing*. Microarrays are popular because of their low price and high throughput, but they can only identify already known sequences. In comparison, Sanger sequencing allowed the identification of novel sequences, but at significantly higher cost and lower throughput.

The invention of *second-generation sequencing* in the early to mid-2000s revolutionized the field of transcriptome analysis. With its high throughput compared to Sanger sequencing, second-generation sequencing allowed to not only identify the transcripts in the sample, but to also quantify their amounts.

The small size of the sequenced fragments (called *reads*) brought its own challenges to the analysis of RNA-sequencing (RNA-seq) data. With the lengths in at most a few hundred bases, the reads had to be *assembled* computationally to form the best-guess prediction of the transcripts and their expression levels in the sample. In the past decade this field has received a huge amount of interest, with countless tools developed for different sequencing platforms (for surveys of tools see e.g. [42, 17, 6, 60, 26, 65]).

In the early 2011 sequencing technologies took a leap forward in the form of *third-generation sequencing* platforms. Compared to the second-generation platforms, third-generation sequencing produces significantly longer reads, up to tens of thousands of bases long, but at a lower throughput and with a higher error rate. In the optimal case, third-generation sequencing would allow us to sequence transcripts in full from one end to

another. But while DNA fragments with lengths in the tens of thousands of bases are routinely sequenced with third-generation platforms [1], RNA sequencing has yet to reach such lengths due to the fragility of the RNA molecule [75, 80, 81]. However, even long reads that do not span the whole transcript can give us valuable information about non-consecutive exons.

The transcript assembly problem can be modeled as Minimum Path Cover problem in a *splicing graph* [27]. In a splicing graph, the vertices correspond to exons and there is an edge between two vertices if there exists a split read alignment where the corresponding exons are consecutive. Then the minimum number of paths covering all vertices corresponds to the most parsimonous solution of assembled transcripts.

Bao *et al.* [5] proposed, and Rizzi *et al.* [67] fully solved, a model of Minimum Path Cover with Subpath Constraints, where the long reads spanning multiple exons create subpaths (also called *exon chains*) that have to be fully contained in some path in the solution.

Thematically, this thesis is divided into two main parts: short-read RNA-seq analysis and long-read RNA-seq analysis. With "short-read" we refer to reads with length of at most a few hundred basepairs (bp) that are produced by second-generation sequencing platforms. "Long-reads" on the other hand can be several thousand bp long, produced either by third-generation platforms or by creating synthetic reads *in silico* from short reads.

In Sections 1.1, 1.2 and 1.3 we will describe the biological background and introduce some notation and algorithms that we will use throughout the thesis.

In Chapter 2 we will discuss the field of short read RNA-seq. In this thesis we assume that we have a reference genome, and the workflow of the RNA-seq data analysis goes from sequencing to read mapping to transcript assembly. Our main contribution to this topic is a novel transcript assembly tool that is based on minimum-cost flows.

We will begin Chapter 3 with an introduction to the third-generation sequencing, and discuss the differences between short and long reads from the perspective of data analysis. All the main contributions of this chapter are related to using long reads as subpath constraints (exon chains) for transcript assembly. In Section 3.3 we survey and experimentally evaluate approaches for optimizing the correctness of exon chains instead of the local alignment score. In Section 3.4 we propose an approach to extend co-linear chaining from two sequences to a sequence and a directed acyclic graph. And in Section 3.5 we describe our implementation of solving Minimum Path Cover with Subpath Constraints.

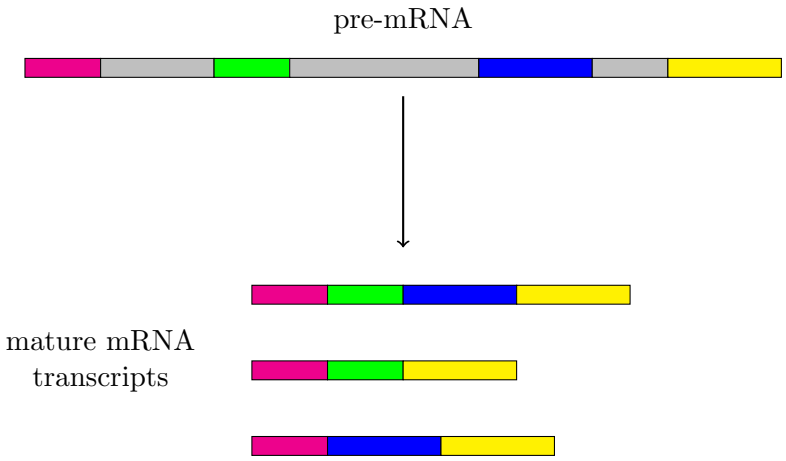


Figure 1.1: During the preprocessing of mRNA, introns (gray boxes) are spliced out and the remaining exons can form different combinations.

Finally, Chapter 4 summarizes the contributions and discusses possible directions for future research.

1.1 Transcription and alternative splicing

The information content of an organism is recorded in its DNA, and expressed through *transcription*. *RNA transcripts* can be divided into two groups, *messenger RNAs* (mRNAs for short) and *non-coding RNAs*. mRNA serves as an intermediary molecule in the production of proteins, whereas non-coding RNAs can perform diverse functions. The collection of RNA transcripts in a cell at any given time is called a *transcriptome*.

After transcription, mRNAs go through a series of modifications to form a mature transcript. The first step in the modifications is the *splicing* out of the introns ("intervening sequences") and the combination of the remaining exons ("expressed sequences"). Because the exons can combine in different ways, a single gene can act as a blueprint for several transcripts. This phenomenon is called *alternative splicing* and is illustrated in Figure 1.1.

The amounts of each RNA transcript in a cell vary based on several factors (e.g. tissue type). When studying the transcriptome, not only do we need to identify the transcripts, but we also need to quantify their amounts.

1.2 Graph theory terminology

A graph $G = G(V, E)$ consists of a set of *vertices* $V = v_1, v_2, \ldots, v_{|V|}$ (also called *nodes*) and a set of *edges* $E \subseteq V \times V$ (also called *arcs*).

A *labeled graph* is tuple (V, E, ℓ, Σ) where (V, E) is a graph and $\ell : V \mapsto \Sigma$ assign labels from Σ to the vertices, Σ being an ordered alphabet.

An edge from u to v is written as $(u, v) \in E$. If there exists an edge (u, v), we call u and v adjacent. If $(u, v) = (v, u)$ we call the edge *undirected*, otherwise we call it *directed*. If all the edges in the graph G are directed, we call G *a directed graph*. In a directed graph, we denote by $N^-(v)$ the set of *in-neighbors* of v and by $N^+(v)$ the set of *out-neighbors* of v. In other words, there exists an edge (u, v) for all $u \in N^-(v)$ and an edge (v, w) for all $w \in N^+(v)$.

A *path* P is a sequence of vertices $v_1 v_2 \ldots v_{|P|}$ such that all consecutive vertices are connected by an edge. We call the first vertex of a path P its *startpoint*, and the last vertex its *endpoint*. If there is a (possibly empty) path from vertex u to vertex v we say that u *reaches* v. We denote by $R^-(v)$ the set of vertices that reach v.

A *cycle* is a path from a vertex to itself. If the graph G has no cycles, it is called *acyclic*. *A directed acyclic graph* (DAG for short) is one of the core components used in this thesis.

A *path cover* of a DAG $G = (V, E)$ is a set of paths $P_1, \ldots, P_k$ such that every vertex $v \in V$ appears on some path. The size of a minimum path cover of G is called the *width* of G.

1.3 Flow networks and minimum-cost flow

A *flow network* is a tuple $N = (G, b, q)$ where $G = (V, E)$ is a directed graph, b is a function assigning a *capacity* $b_{uv} \in \mathbb{N}$ to every edge $(u, v) \in E$, and q is a function assigning an *exogenous flow* $q_v \in \mathbb{N}$ to every vertex $v \in V$, such that $\sum_{v \in V} q_v = 0$. Moreover, we also require that for every source s, we have $q_s > 0$, for every sink t, $q_t < 0$ and for every other node v that $q_v = 0$.

We say that a function x assigning to every edge $(u, v) \in E$ a number $x_{uv} \in \mathbb{N}$ is a *flow* over the network N, if the following two conditions are satisfied:

1. $0 \leq x_{uv} \leq b_{uv}$, for every $(u, v) \in E$,
2. $\sum_{u \in V} x_{vu} - \sum_{u \in V} x_{uv} = q_v$, for every $v \in V$.

In a minimum-cost flow problem, we are additionally given flow cost functions $c_{uv}(\cdot)$, for every edge $(u, v) \in E$. The goal is to then find a flow which minimizes:

$$\sum_{(u,v)\in E} c_{uv}(x_{uv}).$$

1.4 Main contributions

The main contributions of this thesis are given in the original publications I-V. Below is a brief summary of the main results in the order of original publication.

Paper I. In this paper we introduce a method for transcript assembly (identification and quantification) that is based on minimum-cost flows. The method works by finding the minimum-cost flow in an offset flow network in polynomial time. The resulting flow can then be split back into paths by e.g. repeatedly removing the path of maximum bottleneck.

The author carried out the experiments and wrote the corresponding section of the paper.

Paper II. In this paper we refine the problem introduced in Paper I by asking for a number k of paths to optimally explain the splicing graph. Although this problem then becomes NP-hard, we give a fast dynamic programming algorithm for it. We show that this approach, augmented with three optimizations and heuristics, achieves similar or better performance than state-of-the-art tools Cufflinks, IsoLasso and SLIDE.

The author carried out the experiments and wrote the corresponding section of the paper.

Paper III. We implement the concept of using subpath constraints in solving the minimum path cover problem [5, 67] for transcript assembly, and compare this approach to two state-of-the-art tools StringTie and FlipFlop. We use the conditions of hypothetical perfect (error-free) sequencing, and show that even under these conditions the problem is not trivial. Introducing errors to the data complicates the problem further.

The author designed and carried out the experiments and co-wrote the paper. Implementation was joint work.

Paper IV. In this paper we survey and experimentally compare various approaches for finding exon chains corresponding to long read alignments. We also study the time and memory requirements of various modules required for finding the exon chains.

Our experiments show that using short reads from second-generation sequencing can significantly improve the correctness of the predicted exon

chains. Using short reads to error correct the highly erraneous long reads before splicing graph creation, or creating the splicing graph from the short reads and projecting the long reads on the graph show the most improvement based on our experiments.

Design of the experiments was joint work. The author was responsible for carrying out the experiments and writing majority of the paper.

Paper V. In this paper we generalize the dynamic programming solution for co-linear chaining from between two sequences to between a sequence and a directed acyclic graph (DAG) using path covers. The time bounds for our solution reduce to the optimal time bounds for two sequences when the graph corresponds to an unary path.

Our solution is based on considering each path of the path cover as a sequence, combined with careful bookkeeping to keep the computation order correct.

The author co-designed the algorithms and co-wrote the paper.

Chapter 2

Short-read RNA-seq analysis

In the early to mid-2000s *Sanger sequencing* [71], which had been used to compose the first draft of the human genome [32, 87], was eclipsed by *second-generation sequencing* technologies that had much higher throughput with lower cost.

Several companies entered the market within a few years, each with a different technology. 454's pyrosequencing yielded reads of up to 1,000 bp, whereas Illumina's sequencing-by-synthesis approach started out at 36 bp read length, but has grown over the last decade to 300 bp [21]. The differences in the sequencing approaches had an impact on the error profile of the reads as well; while both platforms have ~1% error rate, for 454 the dominant error type is *insertions* and *deletions* (a base being added or removed, *indels* for short), while Illumina's errors are mostly *substitutions* (a base being replaced by a different base).

Other major players in the second-generation sequencing field are Ion Torrent and Applied Biosystem's SOLiD [21]. Ion Torrent's error profile is similar to 454's, with insertions and deletions dominating. The read length is shorter, with 400 bp being the maximum in the current machines. SOLiD sequencing interrogates every base three times by overlapping the primers, which reduces the error rate to under 0.1%, but the drawback of the system is the read length being limited to 100 bp.

Despite the myriad options for second-generation sequencing, research is increasingly being conducted on Illumina instruments [21]. Because of this, we will focus on Illumina-type short reads.

We will first briefly describe the second-generation sequencing process, and then discuss the downstream analysis. We assume that we have a reference genome for the analysis. A different line of research is focused on reference-free transcript assembly (e.g. [10, 73, 22, 94]).

A common reference-based RNA-seq analysis workflow is shown in Figure 2.1. Section 2.2 of this chapter describes the first step, short read mapping, and Section 2.3 focuses on the following transcript assembly step, briefly discussing transcript assembly tools in general before introducing our tool Traph. Differential expression analysis is a vast field with enough material to fill several theses, but is outside the scope of this work.

2.1 Second-generation sequencing

Second-generation RNA sequencing consists of two steps: library preparation and sequencing. As the sequencing platforms were designed for double-stranded DNA, RNA must first be reverse-transcribed into double-stranded *complementary DNA* (cDNA). Reverse transcription can create biases in the resulting library due to the nature of the process (e.g. random hexamer primers show a clear sequence-specific bias [25]). Either before or after reverse transcription, the RNA (or the resulting cDNA) is broken into short fragments.

Optionally the fragments can be duplicated with polymerase chain reaction (PCR) to achieve desired *coverage*, i.e. how many reads cover each base of the transcriptome in average. This step can add its own biases, as not every fragment is copied equally many times [92]. An example of real RNA-seq data aligned to the reference genome is shown in Figure 2.2 to illustrate that these biases cause the coverage to be far from uniform.

After the library preparation, each fragment is clonally amplified to form a cluster, that consists of ~1,000 copies of the fragment. Depending on the sequencing machine (e.g. MiSeq, NextSeq), each sequencing run can accomodate from tens of millions to a few billions clusters, divided between several flowcells. The fragments can be sequenced either from one end of the fragment (*single-end* reads) or from both ends (*paired-end* reads).

During sequencing, nucleotides (A, C, G and T) tagged with fluorescent dyes are added into the flowcells. Because of the dyes' relatively large size blocking the DNA polymerase, only one complementary nucleotide should incorporate to the chain before the reaction terminates. After the excess nucleotides are washed off, the fluorescence from the incorporated nucleotides can be imaged. Cleaving off the dye allows the reaction to continue for another round of sequencing, up until the desired read length.

The "short read" nature (max 2x 300 bp paired-end reads with Miseq v3 chemistry kit [21]) of Illumina systems is because of the nature of the sequencing chemistry. At any point, a cluster can go "out of sync" – for example one of the 1,000 copies may accidentally incorporate two bases

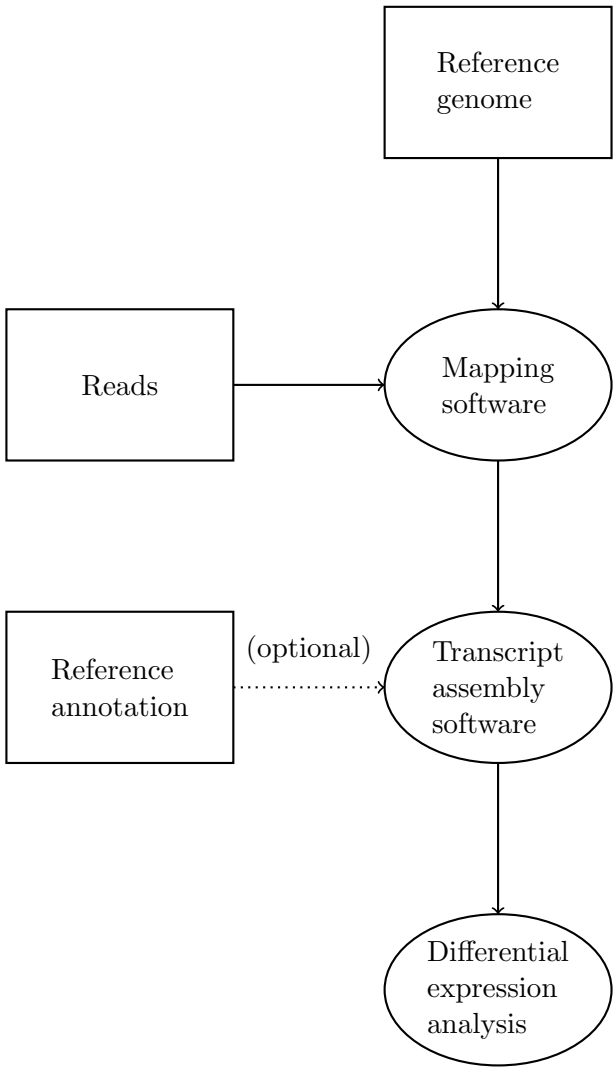


Figure 2.1: Common steps of a reference-based RNA-seq analysis are read mapping, transcript assembly and differential expression analysis.

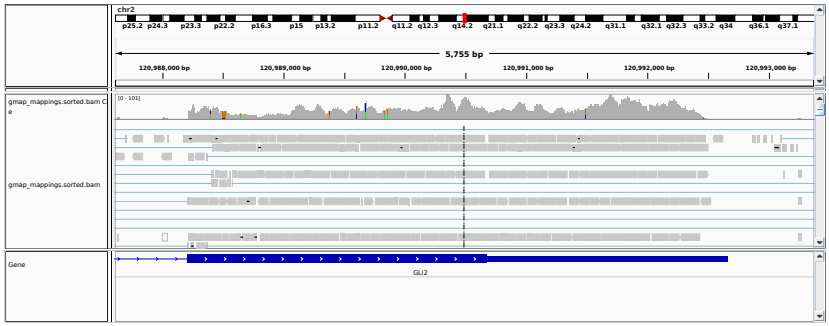


Figure 2.2: An example of real RNA-seq data aligned to the reference genome shows that the coverage is far from uniform. The grey peaks at the top of the image show the coverage of each base, and the grey boxes below represent individual alignments. The blue boxes at the bottom represent the annotated transcripts. Image captured from Integrative Genome Browser [68].

instead of one. Now 999 copies of the fragment give the correct signal and one gives the wrong signal because it is out of sync. The more cycles there are, the more molecules will be out of sync, and eventually the noise from the unphased signal will drown the true signal.

2.2 Short read mapping

The first step of the downstream analysis is to map the reads to a suitable reference. This first step is critical, for the accuracy of the downstream analysis depends on it. While finding the optimal solution for mapping one read to a reference is trivial using dynamic programming, mapping up to billions of reads that way is not feasible in the lifetime of the universe [1].

For faster processing, mapping algorithms construct *indexes* from the read sequence, or the reference sequence, or sometimes both [42]. Although with the current throughput of the second-generation sequencing technologies, approaches that index the reads have become inpractical due to the memory requirements. There are two major indexing approaches, depending on the properties of the index: those based on hash tables (e.g. [15, 45, 34, 43]) and those based on compressed prefix or suffix array-like structures (FM-index) [19] (e.g. [37, 91, 31, 41]).

[1] With the human genome consisting of three billion bp, and assuming read length of 100 bp and filling each cell of the dynamic programming matrix taking one millisecond, mapping one billion reads would take 95,000,000,000 years

Hash table based approaches store into a hash table all k-mers, for some suitable value of k, in the reference. They then query the k-mers in the read for a quick seeding of alignment candidates, which can then be extended or discarded using variations of dynamic programming algorithms [49]. FM-index requires significantly less space than hash tables (e.g. Bowtie's [38] index for the human genome requires only 2.4 GB) and supports extremely fast retrieval of exact matches. As the FM-index is conceptually equivalent to a suffix tree, exact substring matching corresponds to finding a path representing the query, starting at the root. As such, FM-index-based mapping algorithms can query for the exact match of the whole read in one pass.

Mismatches in the alignment can be introduced to hash tables by using e.g. *spaced seeds* [55], i.e. seeds where specified positions are allowed to not match. However, multiple seed masks are required to cover various permutations of match and mismatch positions, which increases the memory requirements. The FM-index allows mismatches at no extra memory cost by branching the search, but the time requirement increase is exponential in the number of allowed mismatches.

Compared to mapping DNA reads that generally align in full with a small number of errors, mapping RNA-seq reads poses an additional challenge; we have to allow for very long gaps in the alignment to account for the spliced-out introns. In mammalian genomes, introns span a wide range of lengths, typically from 50 to 100,000 bases. Most current mapping software for RNA-seq reads use the *seed-and-extend* approach. In the seed-and-extend approach partial reads ("seeds") are matched to the reference genome to find candidate positions ("hits") and then the sequence surrounding the hits is further examined ("extension") to compose a whole read alignment.

2.3 Transcript assembly and Traph

The second step in the downstream analysis of RNA-seq data is transcript assembly, namely reconstructing as accurately as possible the RNA transcripts from the read alignments. The difficulty of the problem lies in the fact that the transcripts may share exons. Because of this, all methods for solving the problem use graph models (either explicit or implicit). The vertices of the graph represent individual reads (overlap graph [83]) or continuous stretches of DNA (splicing graph [27, 48, 44] or connectivity graph [18, 47, 24]), whereas edges are inferred from overlaps or spliced alignments.

Every vertex and edge has an associated observed coverage, and the biological problem of reconstructing the RNA transcripts translates to covering the graph with paths. Some of the popular approaches include exhaustively enumerating over all possible candidate paths and finding the minimum cost solution under some cost model (e.g. a least sum of squares [47] or a least sum of absolute differences [48]), and modeling the problem as *Minimum Path Cover* [83, 78, 5] (MPC).

Here we will introduce our novel transcript assembly tool Traph (for Transcripts in gRAPHs): how Traph creates the splicing graph, how it uses minimum-cost flows and how it splits the resulting flow into paths that correspond to the transcripts. We will also introduce a variant of Traph where, instead of searching for a solution that minimizes the objective cost function, we search for a given number k of paths. Finally we will review the experimental results from comparing Traph to state-of-the-art tools. The material here is based on Paper I and Paper II.

2.3.1 Creating the splicing graph

The approach used by Traph's splicing graph creation module is *annotation-free*, that is, it does not use the annotated (known) transcripts to guide the creation of the splicing graph. The starting point of the algorithm is examining the coverage $cov(i)$ (i.e. the number of reads that cover that position) at every genomic position i. It then classifies positions where $cov(i) > 0$ as *exonic* and positions where $cov(i) = 0$ as *intronic*. However, because of the errors in the read alignment step, there might be spurious regions where $cov(i) > 0$. We use a heuristic to prune these areas by requiring that for a region to be designated exonic either the length of the region has to be above a given threshold, or $cov(i)$ has to be above a given fraction of the average coverage[2].

It is possible that two exons overlap in genomic coordinates. As a post-processing step for exon finding we split any overlapping exons into pseudoexons as illustrated in Figure 2.3. Pseudoexons, unlike exons, cannot have any overlap between them, which simplifies the next steps.

The edges of the splicing graph are inferred from split-read alignments; if two exons are consecutive in a read alignment, an edge is added between the corresponding vertices. Each vertex v has a coverage $cov(v)$ that equals the average coverage of the corresponding exon (i.e. the sum of reads that cover each of the positions in the exon divided by the exon length), and each edge has a weight $cov(u, v)$ that equals the number of split read alignments

[2]The average coverage is computed by counting all the alignments in the file before starting the creation of the splicing graph.

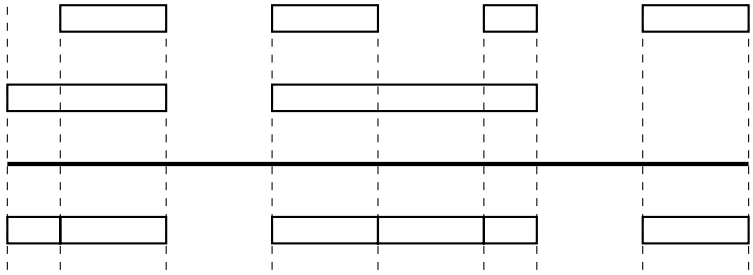


Figure 2.3: During the splicing graph creation we split any overlapping exons (top) into "pseudoexons" (bottom).

corresponding to that edge. Optionally, edges without sufficiently high coverage can also be pruned.

One task remains before moving on to the next step: the flow network requires *flow sources* (vertices with exogenous flow > 0) and *flow sinks* (vertices with exogenous flow < 0). We choose as flow sources the plausible starting vertices of the transcripts, and as flow sinks the plausible ending vertices of the transcripts. As we do not impose any criteria on the number of paths, without given sources and sinks we might have a solution made up of one path per edge, with weight equalling that edge's coverage, which thus has cost 0.

The first heuristic for identifying the flow sources and sinks checks the incoming and outgoing edges of each vertex; if a vertex v has no incoming edges, it is designated a flow source, and if it has no outgoing edges, it is designated a flow sink. The second heuristic examines the coverage profile along the exons to find flow sources and flow sinks featuring alternative splicing events where the flow source (respectively flow sink) of a transcript is contained in an exon of another transcript.

Because the start position of the read along the transcript is random, the theoretic coverage profile of the source vertices and sink vertices is different than for vertices that are in the middle of a transcript (see Figure 2.4). However, due to the biases in the sequencing process, (explained in Section 2.1), the coverage profile in reality is less-than-uniform. We solve this problem by using a sliding window approach; we move a sliding window of size m (default: $m = 10$) along the coverage profile and examine the coverage difference between the positions at start and end of the window.

An example on the main steps of the module – inferring the exons, inferring the edges, and choosing the sources and sinks – is shown in Figure 2.5.

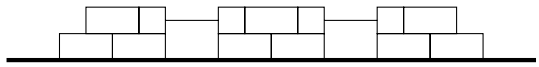


Figure 2.4: With the assumption that the read start positions along the transcript follow an uniform distribution, vertices at the start and end of a transcript have a characteristic slope in coverage.

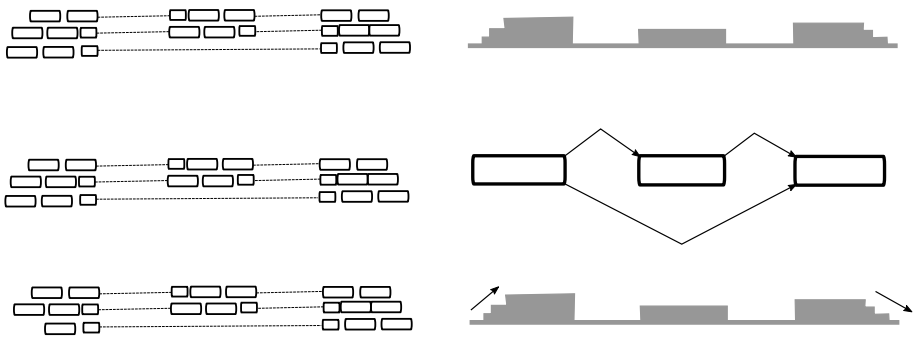


Figure 2.5: Traph's module for creating a splicing graph consists of three main parts: Inferring the exons (top), inferring the edges (middle), and choosing the sources and sinks (bottom). Exons are inferred by examining the coverage profile (on the right). Edges are inferred from the spliced alignments. Flow sources have a characteristic upward slope in coverage, whereas flow sinks have a characteristic downward slope.

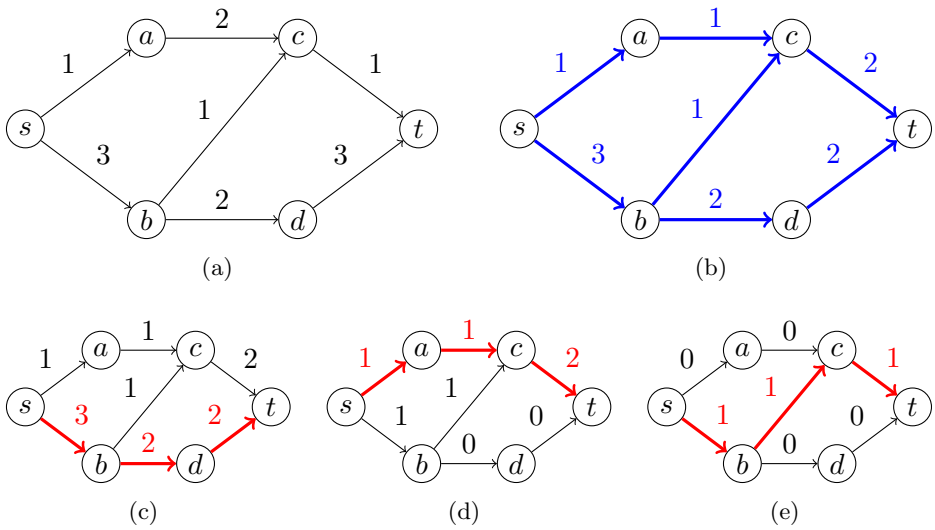


Figure 2.6: Figure 2.6(a): The input graph, where the numbers denote the vertex and edge coverages. We assume a cost function $f(x) = x^2$. Figure 2.6(b): An optimal flow with cost $(1-1)^2 + (2-1)^2 + (1-2)^2 + (3-3)^2 + (1-1)^2 + (2-2)^2 + (3-2)^2 = 3$. Figures 2.6(c) to 2.6(e): The greedy decomposition algorithm iteratively reports the path of maximum bottleneck (red) and then removes those flow values from the network.

2.3.2 The flow engine

First, to transform the splicing graph into an input graph for finding minimum-cost flow we need to do a few simple changes:

1. For every vertex v we replace v with two vertices v_{in} and v_{out}.
2. We add an edge (v_{in}, v_{out}) with weight $cov(v_{in}, v_{out}) = cov(v)$.
3. We add a global source s_0 and a global sink t_0.
4. We add an edge (s_0, s) for every vertex $s \in$ flow sources, and an edge (t, t_0) for every vertex $t \in$ flow sinks.

For finding the optimal flow in polynomial time we used the LEMON library [39]. After finding the optimal flow, the final step is decomposing it into a small number of paths. Decomposing a flow to the minimum number of paths is an NP-hard problem [85, Proposition 2], but there are polynomial-time heuristics for it. We used a greedy approach, where we

repeatedly removed the path of maximum *bottleneck*, where the bottleneck is the smallest flow value of its edges. We then reported it as a transcript, and removed the value of the path from the flow values of all edges it traversed in the network. An example of an input graph, choosing an objective function $f(x)$, finding the optimal flow and decomposing the flow using greedy algorithm is shown in Figure 2.6.

Traph, unlike Traphlor that is introduced in Section 3.5, also produces the expression level estimates for the predicted transcripts. When decomposing the minimum-cost flow into paths, we also implicitly obtained a value for every path; these can be converted into the standard measurement of expression levels: fragments per kilobase of exon per million reads mapped (FPKM).

2.3.3 Dynamic programming – bounding the number of paths

As a follow-up to Paper I, in Paper II we introduced a case where instead of searching for a solution with an arbitrary number of paths, which minimizes the objective cost function, we search for a given number k of paths. Solving this variant of the problem requires $O(|M|^k(n^2 + \Delta^k)n^k)$ time, where M is the set of all possible expression levels, Δ is the maximum in-degree of the graph and n is the number of pseudoexons.

This solution is clearly intractable as-is for large values of k. However, we can use three heuristics and optimizations to make the approach practical. First, we use dynamic programming; we decompose the graph along vertices whose removal disconnects the graph into components and solve the problem recursively on each. Obviously the cut vertex between subgraphs G_1 and G_2 is the sink of G_1 and the source of G_2, so the solution on G_1 can be used to initialize the dynamic programming table for G_2.

Given k paths and the set M of all possible expression levels, enumerating over all the combinations of expression levels for all the paths requires $|M|^k$ time. We avoided this enumeration by using a genetic algorithm. While by its nature the results of a genetic algorithm vary, we experimentally showed that the variation is very small; in one hundred executions, the standard deviation of the path weight was less than 0.001% of the mean path weight for each transcript.

As the time requirement is exponential on k, for practical purposes we implemented a heuristic that chooses k' based on the size of the graph (for $k' \leq k$). We compute the optimal k' paths, remove them from the graph, and repeat the process till we have obtained k paths in total.

The details of the algorithm and the proofs of NP-hardness of the problem can be found in Paper II.

2.3.4 Experimental results

Here we will summarize the experimental results from Paper I and Paper II. In both papers we used simulated data from human chromosome 2; in Paper II we used all the genes that had transcripts longer than 300 bp (1,462 genes), whereas in Paper I we used a subset of them (29 genes) due to the time constraints for submission.

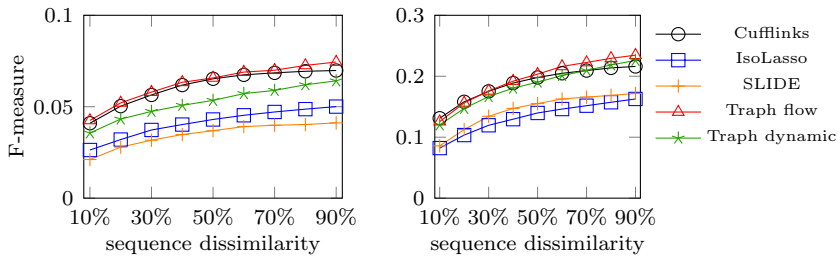
We considered two cases which we call `singles` scenario and `batch` scenario. In the `singles` scenario we simulated 300,000 paired-end reads with length 2x 76 bp from each gene (with the simulated expression levels of the transcripts varying over three levels of magnitude) and gave them to TopHat [82] for alignment one data set (gene) at a time. In the `batch` scenario we pooled those 300,000 paired-end reads per gene for all the simulated genes (1,462 and 29, respectively) and aligned all 438,600,000 and 8,700,000, respectively, reads at once. In the `singles` scenario we strived for idealistic alignments with very little to no noise, whereas the `batch` scenario served as a more realistic alignment setting.

For validation, in both Paper I and Paper II we created a bipartite graph with one side having a vertex for each annotated transcript (from which reads were simulated), and one side having a vertex for each predicted transcript. Dummy vertices, with empty sequence and expression level of 0, were added to the side with less vertices to make the counts even. Each vertex was connected to all the vertices in the opposite side of the graph, with the weight of the edge computed from the differences of the sequences and expression levels. Note that this matching was done on the "per gene area" basis to limit the number of vertices, as we could not run the validation script in a reasonable time for the whole `batch` mode data set.

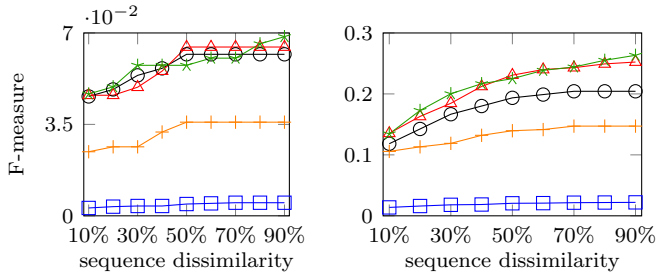
However, there was one major difference in the validation criteria between the papers. In Paper I we used *bitscore* between the sequences, which is based on normalized compression distance [14], whereas in Paper II we used *sequence dissimilarity*. Bitscore is more grounded as a measure, but it requires full alignment. Sequence dissimilarity can be computed with Myers's bitparallel algorithm for approximate string matching [61], which allowed us to use a data set that was two orders of magnitude larger.

Given the bipartite graph with the edge weights computed as described above, we computed a perfect matching where each predicted transcript (or a dummy) matched exactly one annotated transcript (or a dummy). From this matching we then computed true positives, false positives and false negatives under specified sequence and expression level differences[3].

[3]Transcript pairs with sequence and expression level difference under the threshold



(a) `singles` expr. diff. $\leq 10\%$ (b) `singles` expr. diff. $\leq 40\%$



(c) `batch` expr. diff. $\leq 10\%$ (d) `batch` expr. diff. $\leq 40\%$

Figure 2.7: Performance of the tools on simulated data. Plots 2.7(a) and 2.7(b) depict results in the `singles` scenario, and plots 2.7(c) and 2.7(d) depict results in the `batch` scenario. Figures reproduced from Paper II.

We measured the performance between our flow-based Traph ("Traph flow"), Traph with bounded number of paths ("Traph dynamic"), Cufflinks [83], IsoLasso [47] and SLIDE [44] using *F-measure*, the harmonic mean of sensitivity and precision, as the measure of goodness.

Here we show only the results from Paper II as the results from Paper I are included within them. We chose two thresholds for the relative expression level differences, namely 10% and 40%. As can be seen in Figure 2.7, in the `singles` scenario, both Traph flow and Traph dynamic are on-par with Cufflinks and outperform IsoLasso and SLIDE. The situation is the same in the `batch` scenario when the relative expression level difference is at most 10%. In the `batch` scenario with the relative expression level difference at most 40% both our methods outperform the competitors.

Note that in the `batch` scenario any transcripts predicted outside the annotated gene areas were added as false positives, whereas in the `singles` scenario they were simply discarded from the counts. As IsoLasso pre-

were classified as true positives. Otherwise they were classified as false positives or false negatives, depending on which side of the bipartite graph they resided. Under this validation model true negatives do not exist.

dicted a surprising 7,422 transcripts outside the gene areas, this lowered its performance in the `batch` scenario significantly.

Following the publication of Paper I and Paper II, Traph has been used as one of the comparison software in three publications [59, 64, 50]. As expected, the newer software tools outperformed Traph. However, Traph's performance was comparable to Cufflinks and IsoLasso in several of the tests on simulated data. All three of the articles reported that the authors had been unable to run Traph on the real data, which is unfortunate as we had not experienced such problems ourselves and could probably have fixed the issue once contacted.

Chapter 3

Long-read RNA-seq analysis

In the early 2011 the first *third-generation sequencing* platform PacBio RS from Pacific Biosciences was commercially released. It was followed in a few years by Oxford Nanopore's MinION, as well as PacBio's new Sequel System with much greater capacity than PacBio RS.

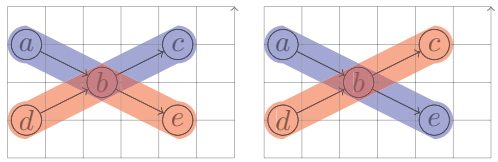
Compared to second-generation sequencing, third-generation sequencing reads are significantly longer (up to hundreds of thousands of bases), but the raw error rate is also an order of magnitude higher [21, 53]. As a response to the high error rates of third-generation sequencing, *synthetic long read* technologies (e.g. Illumina's TrusSeq) have also emerged. These synthetic approaches, first developed by Moleculo[1], combine a novel library preparation step with proprietary informatics to assemble long reads *in silico* from short, second-generation reads.

With the read lengths exceeding the length of all but the longest of transcripts, it would seem that long-read technologies make the RNA-seq analysis trivial, as the transcripts could be sequenced in full from one end to another. However, the library preparation steps for RNA-seq restrict the lengths of the long reads, making the recovery of full-length transcripts for longer genes less likely [75, 80, 81].

Even for the cases where we are not able to recover full-length transcripts, long reads can still provide valuable information about the connectivity of non-consecutive exons (see Figure 3.1 for a simple case where connectivity information of only consecutive exons is not enough).

Long reads, in the form of expressed sequence tags (ESTs), had been used in transcript prediction long before the era of third-generation sequencing [20]. However, in the case of [20] the candidate transcripts were created from exhaustively enumerating over all the paths in the splicing

[1] Illumina acquired Moleculo in late 2012.



(a) (b)

Figure 3.1: A minimal example of a case where the real transcript structure, case (a), is impossible to infer with certainty when given only information about consecutive exons. Case (b) would be equally likely prediction.

graphs, and the ESTs were used to filter for candidates suitable for gene annotation.

Instead, the model of Bao *et al.* considers the long reads spanning multiple exons as *subpath constraints* in a splicing graph[2] [5]. This concept can then be combined with the MPC problem to create the *Minimum Path Cover with Subpath Constraints* (MPC-SC) problem. As the vertices of a splicing graph are exons, a subpath constraint (a sequence of exons) can also be considered an *exon chain*.

In this chapter we will first give an overview of the different options for long-read sequencing, and what trade-offs these technologies offer. Then we will briefly review the literature of aligning long reads to the reference genome as well as describe our experimental study about optimizing the correctness of the exon chains instead of the alignment score, before continuing on to an alternative approach of aligning reads to a splicing graph instead of a reference genome. The chapter will wrap up with the description of Traphlor, our extension to Traph that uses the subpath constraint information.

3.1 Long-read sequencing

There are currently two main technologies that produce long reads: third-generation sequencing and synthetic approaches that use existing short-read technologies to construct long reads *in silico* [21]. Current commercially-available third-generation sequencing technologies can be further divided into two categories: single-molecule sequencing-by-synthesis and nanopore sequencing. Various novel approaches using direct imaging have been attempted (e.g. ZS Genetics, Reveo and Halcyon Molecular) [72], but none of them have been commercially published to this date.

[2]The problem was fully solved by Rizzi *et al.* [67]

Pacific Biosciences' (PacBio) Single-Molecule Real Time (SMRT) sequencing belongs to the sequencing-by-synthesis category. It captures sequence information during the DNA replication process, similar to Illumina sequencing. The main differences are that only a single molecule is sequenced instead of a clonal cluster and that PacBio does not require a pause between the read steps, which makes the sequencing process significantly faster [72].

For PacBio reads, the N50 measure is over 20,000 bp (that is, half of all data are in reads longer than 20,000 bp), with the maximum read length being over 60,000 bp [1]. However, the throughput is low, typical throughput of Sequel System SMRT cell is only 3.5-7 Gb per run, whereas Illumina's HiSeq 2500 produces up to 300 Gb per run. Another disadvantage of PacBio is the error rate, which is roughly 11-15% [3] [66, 53]. The majority of the errors are insertions, followed by deletions.

As the PacBio template – created by ligating two hairpin adaptors to the ends of a double-stranded DNA molecule – is circular, the fragment can be sequenced multiple times (called "passes") over the lifetime of the DNA polymerase [66]. By cutting the adaptors out of the long read, a *circular consensus sequence* (CCS) can be created from the resulting subreads. As PacBio sequencing errors are randomly distributed, unlike the second-generation sequencing errors, the final error rate can be highly reduced by using CCS. For example, a coverage of 15 passes yields $> 99\%$ accuracy. However, as the total length of the long read is limited by the lifetime of the polymerase, the number of passes and the length of the CCS are a trade-off.

The second major player in the third-generation sequencing field is Oxford Nanopore with MinION and PromethION devices [21], which, as the name implies, use nanopore sequencing. Unlike other sequencing platforms, which monitor incorporation of the nucleotides to the template DNA, nanopore sequencing directly detects the DNA composition.

In the nanopore sequencing, a current is passed through a protein pore. As the DNA molecule travels through the pore, the current is disturbed [33]. The shifts of voltage, that are due to the DNA fragment in the pore, can be interpreted as a k-mer corresponding to the DNA fragment currently in the pore. Because nanopore measures k-mers (of length 3-5 bp) instead of single bases, instead of 4 signals, there can be more than 1,000 signals [21].

Oxford Nanopore also uses a hairpin adapter but, unlike PacBio's circular sequence, only ligates it to one end of the DNA molecule [33]. This

[3] Since the accuracy metrics are computed from alignments of base-calls to the appropriate reference, each alignment method used will produce slightly different estimates.

allows forming a two-pass consensus sequence called 2D. An alternate library preparation protocol, 1D, does not use hairpin adapters. The 1D protocol allows for higher throughput, at the cost of lower accuracy.

Nanopore read lengths are even longer than PacBio's; 1D reads over 300,000 bp in length, and 2D reads up to 60,000 bp long, have been achieved with genomic DNA using MinION [33]. The throughput of MinION is a few Gb per run, at the same level as PacBio. But Oxford Nanopore's PromethION, released in late 2016, promised to increase the throughput to 2-4 Tb by massive parallelization and increase in the sequencing speed.

As a consequence of having over 1,000 different signals, Oxford Nanopore has a large error rate: up to 30% for a 1D read and 12% for a 2D read, with insertions and deletions dominating [21]. Homopolymers pose an additional challenge to the platform, as it can be difficult to tell when one k-mer leaves the pore and another one enters.

Illumina and 10X Genomics synthetic long-read approach were developed as a response to the costs ($750-1000 per Gb), error rates (11-30%) and low throughput (a few Gb) of the third-generation sequencing platforms [21]. For creating synthetic long reads, large fragments[4] are partitioned before shearing, and the small fragments in each partition are barcoded.

Then existing short-read sequencing infrastructure can be used for the sequencing, which saves the labs from needing to buy new machines. Accordingly, the throughput and error profile are identical to the existing machines. However, assembling the long reads requires higher sequencing coverage than a typical short-read project, which increases the costs to the same level as PacBio and Oxford Nanopore devices [21].

However, no matter the sequencing technology, for RNA-seq applications sequencing full-length transcripts is still a challenge due to library preparation factors such as RNA degradation, mechanical shearing and incomplete cDNA synthesis [75, 80]. A derivative of Illumina's synthetic long read technology, SLR-RNA-Seq, was tested against PacBio, and it was found that 61-64% of the long reads from both technologies represented full-length transcripts – that is, extending from the first splice site to the last splice site of an annotated transcript [81]. Note that in this study the authors used PacBio's circular consensus sequences to reduce the error rate to 1-2%, which limits the length of the read.

As a final note about the long-read sequencing, it should be remembered that the transcript expression levels vary over six orders of magnitude [29]. This variance combined with the low throughput of PacBio and

[4] As of 2016, the manufacturers reported fragment sizes of $\tilde{1}$00,000 bp.

Oxford Nanopore MinION means that only the most expressed transcripts are likely to be sequenced. Whether PromethION lives up to its promises of high throughput remains to be seen.

3.2 Long read mapping

There are two main characteristics of the reads that we need to consider in the topic of short read mapping vs. long read mapping: read length and error profile. Whereas second-generation sequencing (Illumina) reads are up to 300 bp long with error rate of ~1%, consisting mainly of substitutions, third-generation sequencing platforms PacBio and Oxford Nanopore produce RNA-seq reads of several kb[5], with error rates up to 30%, mainly indels.

In contrast to simple mismatches, only a few algorithms tolerate indels [49, 35, 23]. With commonly used second-generation sequencing mapping tools, even short (1-3 bp) indels in the reads caused the number of correctly reported alignments (the recall measure) to plummet from 92-98% to 47-82% [35]. These issues have inspired a second generation of read mappers (e.g. [88, 58]), as well as new versions of short read mappers that are better equipped for aligning longer reads with high error rates (e.g. [37, 40, 41, 51]). Although, only BWA-SW [41], BWA-MEM [40] and ALFALFA [88] have been shown to scale well to read lengths up to several kilobases.

Additionally, as an increasing fraction of reads contains a splice site, there are more cases where a read extends by 10 bp or less into one of the exons. These small anchors make accurately mapping the reads a difficult task. If an algorithm uses exact matches of k-mers to narrow down the search space, the small anchor often remains unmapped, and is mapped to the intervening intron in the final stage. This problem has been tackled by two-stage alignment (e.g. [35]), where junctions are inferred from the whole data set in the first pass, and then the alignments are adjusted to the junctions in the second pass.

However, for the subpath application introduced earlier, we do not need the exact alignment around the junctions. We simply need the exon chains corresponding to the subpaths. In the following section we will tackle an alternative approach of optimizing the correctness of the exon chains instead of the local alignment score.

[5]As a reminder, third-generation sequencing platforms can produce read lengths in hundreds of kb, but the library preparation factors limit the length of RNA-seq reads.

3.3 Optimizing correctness of exon chains

In Paper IV we evaluated several approaches for optimizing the correctness of exon chains corresponding to long reads. As baseline we created a splicing graph from a mixture of long and short reads. As the main alternative, we created the splicing graph from only the short reads, and used two different approaches to match the long reads on the graph: using dynamic programming on the graph, and an approximate method of choosing the exons with the most overlap with the genomic alignments. Additionally, we attempted to error-correct the long reads using short reads before creating the splicing graph only from the error-corrected long reads.

For error correction we used LoRDEC [70], for alignment GMAP [93] and for creating the splicing graph SpliceGrapher [69]. The rest of the tasks were performed with in-house scripts.

Given a splicing graph created from short reads, we first transformed it into a *sequence graph* by replacing each pseudoexon with a unary path where its ith vertex is labeled with the ith character of the exon. We also considered the long read as a linear sequence DAG. Then we could use dynamic programming to perform a *DAG-path alignment*, with the minor adjustment that we wanted to find a *semi-local alignment*.

We used a scoring scheme that slightly favored insertions and deletions over mismatches (+4 for match, -3 for mismatch and -2 for both insertions and deletions). These values fall short of the real error profile of PacBio reads (where insertions are approximately ten times more common than substitutions [13]), but a fully realistic error profile, e.g. -10 for mismatch, -1 for insertion and -5 for deletion, would cause every true mismatch to be reported as an insertion followed by a deletion.

As a fast approximation for the dynamic programming, we tested an approach of choosing the exons for the exon chain based on overlaps between the genomic coordinates of the long read alignment and the genomic coordinates of the exons in the splicing graph created from short reads.

For the purpose of the study in Paper IV, we implemented a naive approach where for every long read alignment and for every exon in the splicing graph we calculated the overlap between the coordinates. Every exon e that had an overlap with a read R was considered a candidate for the path. From these candidates we chose n exons that formed a path $P = e_1, \ldots, e_n$, where the start coordinates of the n exons were in ascending order. If there existed no edge between exons e_i and e_{i+1}, the path was considered to fail to align.

For the experiments we chose 100 genes at random from human chromosome 2 (GRCh38/hg38) that had at least two distinct transcripts (i.e.

not sharing all the inner borders) with the minimum length of 1 kb and with the longest transcript being over 3.2 kb long.

For each gene we simulated 10,000 short (75 bp) reads with 1% substitution rate and 1,000 long reads (400,800,1200,1600 bp) with 11% insertion, 4% deletion and 1% substitution rate (experimentally estimated PacBio error profile from [13] rounded to the nearest integer). We also simulated the same data sets without sequencing errors for comparison. For simulation we used RNASeqReadSimulator [46] and an in-house script, as RNASeqReadSimulator only simulates substitutions.

For "ground truth" we converted the simulated read data to alignments and compared the genomic coordinates of the alignments to the exons in the splicing graphs produced by SpliceGrapher. For our base case and the error-corrected case we used the same procedure to get the paths. Dynamic programming and overlap methods produced paths as-is. Two paths were considered to match if they consisted of the same set of vertices.

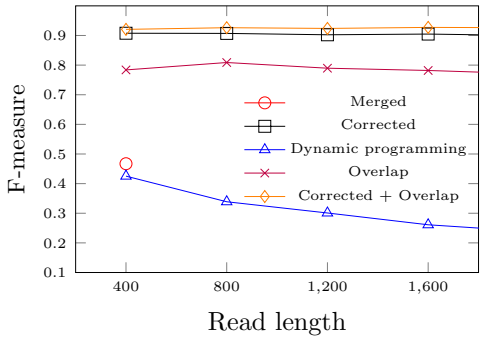
The results for data with 16% sequencing error are shown in Figure 3.2(a) and for the data without sequencing error in Figure 3.2(b). SpliceGrapher for the base case ("merged") could not be run for read lengths longer than 400 bp in reasonable time.

Error correction was clearly the top performer with approximately 90% F-measure. Combining error correction and the overlap methods, that is, creating splicing graph from the short reads, aligning error-corrected long reads to the genome and choosing the exon chains based on the overlaps, improved the performance slightly.

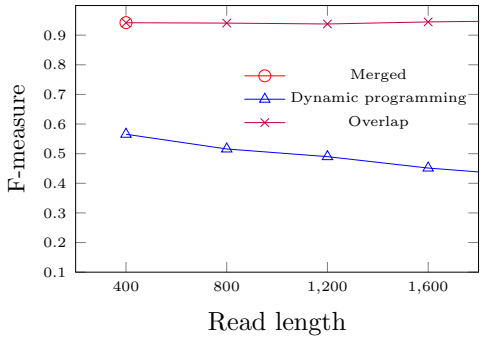
Surprisingly, dynamic programming performed very poorly, with F-measure in the 25-40% range with errors and 45-60% without errors. The likely explanation is that while dynamic programming is guaranteed to find an optimal solution, it can break ties arbitrarily: if the last base of the exon and the last base of the following intron are the same, and both are incorporated into the splicing graph, choosing either will give score-wise optimal solution. But for validation only one of them is correct. This phenomenon is present both with and without sequencing errors.

Read aligners generally use the canonical dinucleotides (AG-GT) to break ties, which explains why the overlap case does not suffer as greatly from the erraneous splice sites being incorporated into the splicing graph.

To study the correlation between the correctness of exon chains and transcript prediction accuracy, we ran tools StringTie [64] and Traphlor (see Section 3.5 and Paper III) to predict the transcripts from the alignments and predicted paths. Traphlor ran on the mixture of short and long reads was considered the baseline (from here on referred to as "Traphlor base").



(a) 16% error



(b) No error

Figure 3.2: Exon chain prediction accuracy for the cases with (a) 16% sequencing error (b) and no sequencing error. In "merged" case long read alignments were mirrored on a graph made from both short and long reads, in "dynamic programming" dynamic programming was used to align long reads on the splicing graph, and in "overlaps" the best overlap in genomic coordinates on the exons predicted from short reads were chosen. In "corrected" the long reads with 16% sequencing error were first error-corrected with short reads, then aligned to the reference and these alignments were mirrored on graph made from short reads. "Corrected + overlap" cases first used error-correction and then used the overlaps between short and long read alignments to infer the exon chains. Figure reproduced from Paper IV.

For the base case we let Traphlor build the splicing graph. For the other use cases (merged, corrected, dynamic programming, overlaps and corrected+overlaps) the graphs from SpliceGrapher and the inferred paths were given to Traphlor's flow engine. StringTie built its own splicing graph.

The running times of Traphlor base and Traphlor's flow engine grew linearly with read length, whereas StringTie's running time seemed to increase exponentially. We aborted the processing of the 1600 bp data set after three CPU days had passed, but in hindsight the expected running time with the exponential increase was almost nine CPU days.

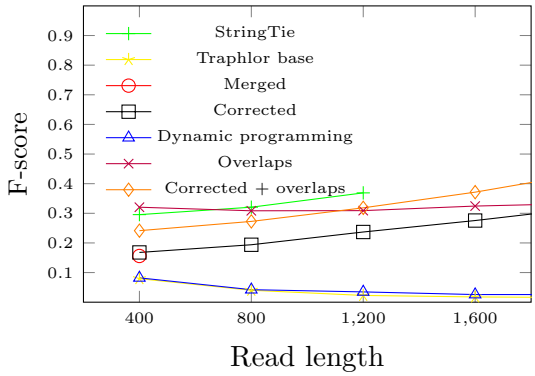
Based on Figure 3.3, our hypothesis – that the correctness of the exon chains correlates with the accuracy of the transcript prediction – seems to hold. Dynamic programming had the worst exon chain accuracy, and is at the same level as Traphlor base in the transcript prediction accuracy. However, the overlap method had slightly worse exon chain accuracy than the error correction method, but slightly better accuracy in the transcript prediction.

3.3.1 Hidden Markov model -based approach to find correct splice sites

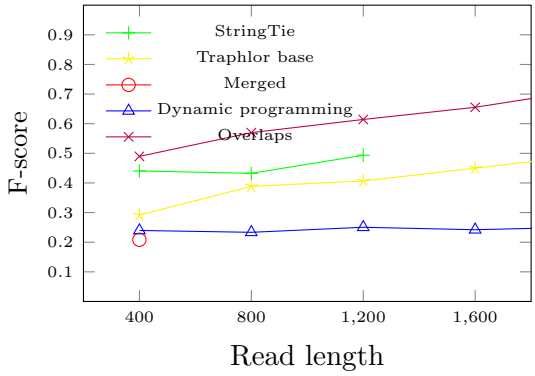
Due to space constraints we had to omit a section about using hidden Markov models (HMM) to correct the splice sites of long read alignments from Paper IV. The main idea behind this approach is to let the canonical dinucleotides, that reside at the borders of exons and introns, guide the search. The pair "GT" at the exon-intron border is called *donor site* and the pair "AG" at the intro-exon border is called *acceptor site*.

Hidden Markov models have been used in several gene prediction software [36, 28, 54, 79], but to our knowledge they have not been used for inferring splice sites from RNA-seq data. As with the gene prediction software, HMM for splice site recognition requires states for "exon" and "intron", as well as "donor" and "acceptor" sites (see Figure 3.4). As the choice of the area to process is based on the read alignments, states for intergenic areas are not needed. The model can also be simplified compared to gene prediction HMM by having only a single exon and a single intron stage, as we do not need to keep track of the reading frame.

Our HMM emits a tuple of symbols at each state: the character at position i and whether the coverage at position i is above or below a given threshold k (see Figure 3.5). The intuition is that there should be at least k amount of coverage in exonic regions, and very little or no coverage in intronic regions. Due to mismappings near splice sites, the coverage near the exon-intron borders could be above or below the threshold.



(a)



(b)

Figure 3.3: Transcript prediction accuracy using the data sets with 16% sequencing error. Transcripts were predicted from the alignment file (both short and long reads) using software StringTie and Traphlor (Traphlor base), which build their own graphs. In the remaining cases the flow network module of Traphlor was given the predicted graphs and exon chains for each exon chain finding approach. Figure reproduced from Paper IV.

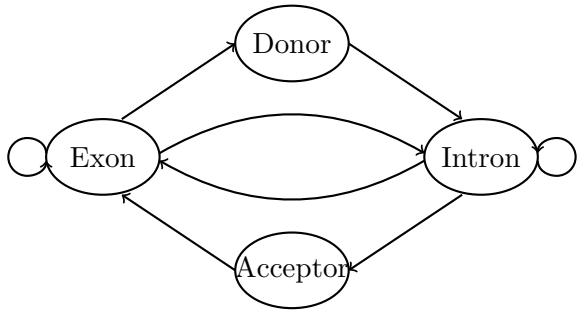


Figure 3.4: A simple hidden Markov model for finding splice sites consisting of four states: Exon, Intron, Donor site and Acceptor site.

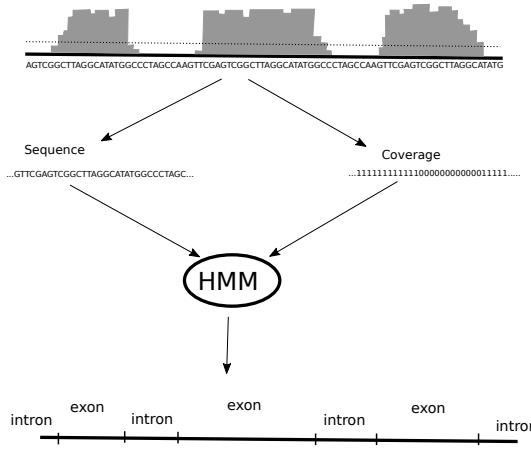


Figure 3.5: From the alignment data we extract the sequence (left) and for every genomic position i also whether the coverage at that position is above or below a given threshold k (right). The tuples (base, coverage) are given to the HMM and Viterbi algorithm is ran to produce a sequence of exonic and intronic areas. Based on these areas the alignments can then be corrected around the splice sites.

Table 3.1: The transition probabilities between the HMM states. The path can start and end in either exon or intron state.

	Exon	Intron	Donor	Acceptor
Exon	0.9	0.01	0.09	0.0
Intron	0.01	0.9	0.0	0.09
Donor	0.0	1.0	0.0	0.0
Acceptor	1.0	0.0	0.0	0.0

Table 3.2: The emission probabilities of the HMM. Emitted symbol is a pair of (base, coverage), where coverage is either 0 (under the threshold) or 1 (above the threshold). Donor and acceptor states can only emit their corresponding dinucleotides.

	(A,0)	(A,1)	(C,0)	(C,1)	(G,0)	(G,1)	(T,0)	(T,1)
Exon	0.0625	0.1875	0.0625	0.1875	0.0625	0.1875	0.0625	0.1875
Intron	0.1875	0.0625	0.1875	0.0625	0.1875	0.0625	0.1875	0.0625

After the publication of Paper IV, we implemented this approach and experimented on the data sets used in Paper IV to see how the input and output alignments differed. For implementation we used the C++ library StochHMM [52].

We tested several combinations of transition and emission probabilities to find values that would result in reasonable exon-intron paths (e.g. not too short exons). The final transition and emission probabilities used in correcting the alignments are listed in Tables 3.1 and 3.2.

Based on the exon-intron path prediction of the HMM, we adjusted the alignments block-wise, where each block was defined as an ungapped part of an alignment. If a first, or respectively last, block crossed a predicted exon border, we attempted to align the crossing part of the sequence to the border of any previous, or respectively following, exon. For the middle blocks we shifted the junction to match the predicted borders, if possible. If the alignment could not be shifted as described above, it was left as-is.

We defined as *correct* an alignment where the junctions of the predicted alignment matched the junctions of the true alignment. Conversely, an alignment that was not correct was classified as *incorrect*. We did not require the outer borders to match to allow for *soft clips*, i.e. unaligned bases at the ends of the alignment.

We measured the total number of modified alignments, as well as how many alignments changed from correct to incorrect, from incorrect to correct, and from one incorrect alignment to another incorrect one.

Table 3.3: The results of modifying read alignments using HMM for Illumina and PacBio type data. The numbers in the parenthesis are the sequencing error rates of the data. The columns show the percentage of the read alignments that were modified, the percentage of correct alignments (C) changed into incorrect alignments (I), incorrect alignments into correct alignments, and incorrect alignments into different incorrect alignments.

	Modified	C→I	I→C	I→I
400 bp				
Illumina (1%)	0.38	14.63	0.03	85.35
Illumina (0%)	0.39	14.61	0.00	85.39
PacBio (16%)	0.39	41.00	0.00	59,00
PacBio (0%)	0.39	23.59	0.00	76.41
800 bp				
Illumina (1%)	0.38	14.11	0.00	85.89
Illumina (0%)	0.39	14.13	0.00	85.87
PacBio (16%)	0.09	19.54	0.00	80.46
PacBio (0%)	0.39	21.52	0	78.48
1200 bp				
Illumina (1%)	0.38	13.93	0.10	85.96
Illumina (0%)	0.39	14.15	0.00	85.85
PacBio (16%)	0.06	16.13	0.00	83.87
PacBio (0%)	0.35	20.80	0.00	79.20
1600 bp				
Illumina (1%)	0.38	13.62	0.05	86.33
Illumina (0%)	0.39	14.28	0.00	85.72
PacBio (16%)	0.06	23.21	0.00	76.79
PacBio (0%)	0.40	25.00	0.00	75.00
2000 bp				
Illumina (1%)	0.38	14.12	0.11	85.77
Illumina (0%)	0.38	14.62	0.00	85.38
PacBio (16%)	0.05	4.17	0.00	95.83
PacBio (0%)	0.47	33.26	0.00	66.74
2800 bp				
Illumina (1%)	0.39	14.46	0.15	85.38
Illumina (0%)	0.40	14.06	0.00	85.94
PacBio (16%)	0.11	9.57	0.00	90.43
PacBio (0%)	0.67	55.13	0.00	44.87
3200 bp				
Illumina (1%)	0.37	13.53	0.08	86.39
Illumina (0%)	0.38	14.08	0.00	85.92
PacBio (16%)	0.06	17.86	0.00	82.14
PacBio (0%)	0.80	44.60	0.00	55.40

Table 3.4: The median running time per gene of using dynamic programming to align 1,000 long reads to the sequence graph. As the time-measuring module used measured real time instead of CPU time, median time is more suitable than mean time to filter out outliers.

400 bp	800 bp	1200 bp	1600 bp
1 hr 30 min	4 hr 55 min	11 hr 50 min	20 hr 34 min

As can be seen in Table 3.3, the number of modified alignments was insignificant for both Illumina (0.37%-0.40%) and PacBio (0.05%-0.80%) type reads. As our validation criteria considered only blocks of contiguous alignment, the correcting of an alignment was not able to "cross" an insertion or a deletion in the alignment, which could limit the number of modified alignments. However, Illumina type reads should have very few indels.

Additionally, at most 0.15% of the modified alignments were changed from incorrect to correct. These results suggest that hidden Markov models might not be a suitable model for this problem.

3.4 Co-linear chaining in DAGs

In Paper IV we evaluated approaches on finding the exon chains corresponding to the long reads in a splicing graph. One of the tested approaches was creating the splicing graph from short, less erroneous, reads, and using dynamic programming to align the long reads to the corresponding sequence graph. As a reminder, a sequence graph is created from the splicing graph by replacing the exons with unary paths where each vertex corresponds to one base of the exon.

However, aligning sequences under edit distance (e.g. filling the dynamic programming matrix) requires quadratic time, and as such is not feasible for large sequence graphs and long reads (see Table 3.4 for some experimental running times from Paper IV).

Co-linear chaining was already a crucial component of whole genome alignment algorithms in the early 2000s [74, 12, 11], but it has also gathered recent interest over the past few years for its scalability to massive inputs (see e.g. [63, 84, 88, 89, 90, 57]).

In the co-linear chaining problem, the input is assumed to be a set of N anchor pairs of intervals in two sequences that match, either exactly or approximately. The goal of co-linear chaining is to find a subset of these plausible anchors whose elements appear in increasing order in both se-

quences and which maximize the coverage in one of the sequences. This type of solution, extended to a DAG, is sufficient for our purpose of finding subpath constraints; the exact alignments at the exon borders are not needed.

Formally, the co-linear chaining problem between a sequence and a DAG can be formulated as follows:

Problem 3.1 (Co-linear chaining between a sequence and a DAG) *Let R be a string, let G be a labeled DAG, and let M be a set of N anchor pairs $(P, [c..d])$, where P is a path in G, $\ell(P)$ is the concatenation of the vertex labels in P, and $c \leq d$ are non-negative integers (with the interpretation that $\ell(P)$ matches $R[c..d]$). Find an ordered subset $S = s_1 s_2 \cdots s_p$ of pairs from M such that*

- *for all $2 \leq j \leq p$, it holds that $s_{j-1}.P \prec s_j.P$ and $s_{j-1}.d < s_j.d$, and*
- *S maximizes the* ordered *coverage of R, defined as* $\texttt{coverage}(R, S) = |\{i \in [1..|R|] \mid i \in [s_j.c..s_j.d] \text{ for some } 1 \leq j \leq p\}|$.

We say that P_1 *precedes* P_2 ($P_1 \prec P_2$) if either P_1 and P_2 have a suffix-prefix overlap (and P_2 is not fully contained in P_1), or they do not share vertices and the endpoint of P_1 reaches the startpoint of P_2.

Let us first study a relaxed version of the problem where overlaps between the paths are not allowed.

Problem 3.2 (Overlap-limited co-linear chaining between a sequence and a DAG) *Let R be a string, let G be a labeled DAG, and let M be a set of N anchor pairs $(P, [c..d])$, where P is a path in G and $c \leq d$ are non-negative integers (with the interpretation that $\ell(P)$ matches $R[c..d]$). Find an ordered subset $S = s_1 s_2 \cdots s_p$ of pairs from M such that*

- *for all $2 \leq j \leq p$, it holds that there is a non-empty path from the endpoint of $s_{j-1}.P$ to the startpoint of $s_j.P$ and $s_{j-1}.d < s_j.d$, and*
- *S maximizes* $\texttt{coverage}(R, S)$.

The basic idea of solving co-linear chaining problem with dynamic programming is to fill a table $C[1 \ldots N]$ where entry $C[j]$ gives the maximum ordered coverage of $R[1 \ldots M[j].d]$ using the pair $M[j]$ and any subset of the pairs from $M[1], M[2], \ldots, M[j-1]$. As the anchor intervals in the sequence can overlap, we need to consider two cases when filling the table: either the intervals do not overlap (let us call it case (a)) or they overlap (let us call it case (b)).

The dynamic programming table can be filled with the following formula:

$$\begin{aligned} C^{\mathtt{a}}[j] &= (M[j].d - M[j].c + 1) + \max_{j' \,:\, M[j'].d < M[j].c} C[j'], \\ C^{\mathtt{b}}[j] &= M[j].d + \max_{j' \,:\, M[j].c \le M[j'].d \le M[j].d} C[j'] - M[j'].d, \\ C[j] &= \max(C^{\mathtt{a}}[j], C^{\mathtt{b}}[j]). \end{aligned}$$

Inclusions, i.e. $M[j].c \le M[j'].c$, can be left computed incorrectly in $C^{\mathtt{b}}[j]$, since there is a better or equally good solution computed in $C^{\mathtt{a}}[j]$ or $C^{\mathtt{b}}[j]$ that does not use them [2].

The challenge when extending the co-linear chaining problem to a DAG is that the topological order of the graph might not follow the reachability order between the paths. In the following sections we will describe two solutions for dealing with this issue: a brute-force approach using depth-first traversal order, and a path cover approach that links the tuples using *forward propagation*.

3.4.1 Brute-force algorithm

Problem 3.2 can be solved with brute force by checking individually, for every tuple j, every tuple j' for which $M[j'].P \prec M[j].P$. To process only the tuples that fulfill the precedence relation, we use graph traversal as shown in Algorithm 1.

First we order the N pairs $M[1], M[2], \ldots, M[N]$ in $O(|E| + N)$ time so that the endpoints of $M[1].P, M[2].P, \ldots, M[N].P$ are in topological order, breaking ties arbitrarily. Then we reverse the edges of G. After this preprocessing step, we can start computing the maximum ordered coverage for the pairs as follows: for every pair $M[j]$ in topological order of their path endpoints for $j \in \{1, \ldots, N\}$ we do a depth-first traversal starting at the startpoint of path $M[j].P$. Note that since the edges are reversed, the depth-first traversal checks only pairs whose paths are predecessors of $M[j].P$. Depth-first search takes $O(|V|+|E|)$ time and is executed N times, for $O((|V| + |E|)N)$ total time.

Theorem 3.1 *Overlap-limited co-linear chaining between a sequence and a labeled DAG $G = (V, E, \ell, \Sigma)$ (Problem 3.2) on N input pairs can be solved in $O((|V| + |E|)N)$ time.*

Input: DAG $G = (V, E)$ and N pairs $M[1], M[2], \ldots, M[N]$ of the form $(P, [c..d])$..
Output: The index j giving $\max_j C[j]$.
Order the N pairs by the topological order of their path endpoints.
`/* Save to end[`i`] the indexes of all pairs whose path ends at` i`. */`
for $j \leftarrow 1$ *to* N **do**
  `end[`$M[j].P.last$`].push(`j`)`;
end
Reverse the edges of G;
for $j \leftarrow 1$ *to* N **do**
  maxcov $\leftarrow M[j].d - M[j].c + 1$;
  $u \leftarrow M[j].P.first$;
  `/* Depth-first search starting from` u`. When meeting a vertex where a tuple ends, update the coverage. */`
  for $w \in DFS(u)$ **do**
    for $j' \in$ `end[`w`]` **do**
      $C \leftarrow -1$;
      if $M[j'].d < M[j].c$ **then**
        $C \leftarrow C[j'] + (M[j].d - M[j].c + 1)$;
      else if $M[j].c \leq M[j'].d \leq M[j].d$ **then**
        $C \leftarrow C[j'] + (M[j].d - M[j'].d)$;
      if $C > maxcov$ **then**
        maxcov $\leftarrow C$;
    end
  end
  $C[j] \leftarrow$ maxcov;
end
return $\text{argmax}_j C[j]$;

Algorithm 1: Brute-force algorithm for co-linear chaining between a sequence and a DAG.

3.4.2 Co-linear chaining between a sequence and a DAG using path cover

For graphs with small width k, we can do significantly better than the trivial algorithm's time requirement $O((|V| + |E|)N)$. Using a minimum path cover to reduce the DAG into a set of k paths and treating these paths as sequences (plus some additional information about the reachability between the vertices), we can solve Problem 3.2 in time $O(k|E| \log |V| + kN \log N)$. However, the performance difference depends on the topology of the graph compared to the number N of the input pairs, if e.g. $k = O(|V|)$ and $N = O(1)$, the brute-force algorithm performs better.

Our approach is based on the following observation: Suppose that we have a problem involving DAGs that is solvable by traversing the vertices in topological order, and that a partial solution at each vertex v is obtainable from all (and only) vertices that can reach v, denoted $R^-(v)$. Also suppose that at each vertex v we need to to query a data structure that depends on $R^-(v)$ and such that the answer $\mathsf{Query}(R^-(v))$ at v is decomposable as:

$$\mathsf{Query}(R^-(v)) = \bigoplus_i \mathsf{Query}(R_i^-(v)). \tag{3.1}$$

In the above, the sets $R_i^-(v)$ are such that $R^-(v) = \bigcup_i R_i^-(v)$, they are not necessarily disjoint, and $\bigoplus$ is some associative operation on the queries, such as min or max. It is understood that after the computation at v, we need to update the data structure, and that after the update we cannot query for a vertex before v in topological order, because it would give an incorrect answer.

Using the above observation, we replace a single data structure with K data structures, where K is the number of paths in the path cover, and perform the operation from (3.1) on the results of the queries to these K data structures.

Our approach consists of three parts:

1. Finding the minimum path cover, that is, the smallest set of paths that cover all the vertices of the graph.

2. Computing the forward propagation links.

3. Solving Problem 3.2 by treating the paths of the minimum path cover as sequences.

To use flows for finding the minimum path cover, we first transform graph $G(V, E)$ into graph $G*$ by replacing each node v with two nodes v^-

and v^+, add an edge (v^-, v^+) and add all in-neighbors of v as in-neighbors of v^-, and all out-neighbors of v as out-neighbors of v^+. Edges of type (v^-, v^+) get demand 1, whereas all other nodes get demand 0. Finally, we add a global source with an out-going edge to every node, and a global sink with an in-coming edge from every node.

Our algorithm finds the minimum path cover by reducing a minimum flow problem to a maximum flow (see e.g. [3]). First we find a feasible flow $f : E \to \mathbb{Z}$, and then we transform it into a minimum feasible flow by finding a maximum flow f' in G in which every edge $e \in E$ now has capacity $f(e) - d(e)$, where $d(e)$ is the demand of the edge. The final minimum flow is then obtained as $f(e) - f'(e)$, for every $e \in E$.

We use an approximate greedy algorithm for finding a path cover in $G*$ whose size is larger than the size of the minimum path cover k by a factor $O(\log |V|)$. This algorithm is based on the classical greedy set cover algorithm (see e.g. [86]): at each step, choose a path that covers the most uncovered vertices. We compute with dynamic programming for each vertex $v \in V$ value $u[v]$, storing the largest number of uncovered vertices on a path starting from v. This takes $O(|E|)$ time. Then we take the vertex v with the maximum $u[v]$, trace back for the optimal path starting from v, and set every vertex on that path as covered. We repeat this process for K paths, for a total time of $O(K|E|)$. Following the proof of [86], because the universe to be covered is V and each possible path in G is a possible covering set, which implies that $K = O(k \log |V|)$, where k is the size of the minimum path cover.

This path cover induces a flow of value $O(k \log |V|)$, which still needs to be reduced to a flow of value k. If we run the Ford-Fulkerson algorithm on $G*$, this means that there are $O(k \log |V|)$ successive augmenting paths, each of which can be found in time $O(|E|)$, for a total time of $O(k|E| \log |V|)$.

Theorem 3.2 *Given a DAG $G = (V, E)$ of width k, the minimum path cover on G can be found in time $O(k|E| \log |V|)$.*

The second part of our approach is computing the forward propagation links using the path cover $P_1, \ldots, P_K$ just computed above. As the answer at v depends on $R^-(v)$, we cannot process the vertices on the K paths in arbitrary order. We define as `last2reach`$[v, i]$ the *last vertex* on path i that reaches v. See Figure 3.6 for an example.

As the complementary operation, we define a set of forward propagation links `forward`$[u]$, where $(v, i) \in$ `forward`$[u]$ holds for any vertex v and index i with `last2reach`$[v, i] = u$. Computing all the forward propagation links takes $O(|E|K)$ time, as shown in Algorithm 2.

Input: DAG $G = (V, E)$ and path cover $P_1, P_2, \ldots, P_K$ of V.
Output: Propagation links (u, v, i) stored as $(v, i) \in \texttt{forward}[u]$.

```
for i ← 1 to K do
    for j ← 1 to |P^i| do
        v ← P^i[j];
        paths[v].push(i);
        index[v][i] ← j;
    end
end
for v ∈ V in topological order do
    for i ∈ paths[v] do
        last2reach[v][i] ← index[v][i];
    end
    for i ∉ paths[v] do
        last2reach[v][i] ← max_{u∈N⁻(v)} last2reach[u][i];
    end
end
for v ∈ V in topological order do
    for i ∈ paths[v] do
        forward[last2reach[v][i]].push(v, i);
    end
end
```

Algorithm 2: Computing the propagation links.

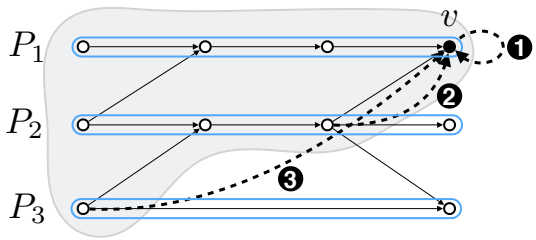


Figure 3.6: A path cover P_1, P_2, P_3 of a DAG. The forward links entering v are shown with dotted black lines, for each of P_1, P_2, P_3. We mark in gray the set $R^-(v)$ of vertices that reach v. Image reproduced from Paper V.

Lemma 3.1 *Let $G = (V, E)$ be a DAG, and let $P_1, \ldots, P_K$ be a path cover of G. For every $v \in V$ and every $i \in [1..K]$, we can compute* `last2reach`$[v, i]$ *in overall time $O(|E|K)$.*

Before continuing on to the third step, we will briefly review an algorithm for the co-linear chaining problem between two sequences that runs in the optimal $O(N \log N)$ time [2], as we will reuse it in our algorithm for co-linear chaining between a sequence and a DAG. We fill follow the notation of [56].

The key component of the algorithm is a pair of binary search trees that support *Range Maximum Queries*: $\mathcal{T}$ that stores the values for cases where the anchors do not overlap, and $\mathcal{I}$ that stores the values for cases where the anchors do overlap.

We will use the following classical result in our analysis:

Lemma 3.2 *The following two operations can be supported with a balanced binary search tree $\mathcal{T}$ in time $O(\log n)$, where n is the number of leaves in the tree.*

- `update`$(k, \texttt{val})$*: For the leaf w with* `key`$(w) = k$*, update* `value`$(w) =$ `val`*.*
- `RMaxQ`(l, r)*: Return* $\max_{w\,:\,l \le \texttt{key}(w) \le r}$ `value`(w) *(*Range Maximum Query*).*

Moreover, the balanced binary search tree can be built in $O(n)$ time, given the n pairs (`key`, `value`) *sorted by component* `key`*.*

Let T and R be two sequences over an alphabet Σ, and let M be a set of N pairs $([x..y], [c..d])$. First we sort the input pairs M by the coordinate

y into a sequence $M[1], M[2], \ldots, M[N]$ so that $M[i].y \leq M[j].y$ holds for all $i < j$. Next we initalize the search trees $\mathcal{T}$ and $\mathcal{I}$ with keys $M[j].d$, for every pair $M[j]$, and additionally the key 0. All the values are set to $-\infty$.

Like in our brute-force approach, described in Algorithm 1, we fill a table $C[1 \ldots N]$ where entry $C[j]$ gives the maximum ordered coverage of $R[1 \ldots M[j].d]$ using the pair $M[j]$ and any subset of the pairs from $M[1], M[2], \ldots, M[j-1]$.

We can use an *invariant technique* to convert the recurrence relations such that we can exploit range maximum queries of Lemma 3.2:

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

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 $\mathcal{T}.\mathsf{update}(M[j'].d, C[j'])$ and $\mathcal{I}.\mathsf{update}(M[j'].d, C[j'] - M[j'].d)$ after computing each $C[j']$. The running time is $O(N \log N)$.

Recall that our approach uses a path cover to split the graph into a series of sequences. Assume we have a path cover of size K and have computed $\texttt{forward}[u]$ for all $u \in V$. For each path $i \in [1..K]$ we create search trees $\mathcal{T}_i$ and $\mathcal{I}_i$. As in the case of sequences described earlier, we set as keys all $M[j].d$ for every pair $M[j]$ and additionally key 0, and initialize all values to $-\infty$.

We process the vertices in topological order. For every vertex v that corresponds to the endpoint of some $M[j].P$, we update the trees $\mathcal{T}_i$ and $\mathcal{I}_i$ for all paths i that cover vertex v. Then we follow the forward propagation links and update $C[j]$ for each path $M[j].P$ whose startpoint is w for all $(w, i) \in \texttt{forward}[u]$.

Before the main loop visits w, we have processed all forward propagation links to w, and the computation of $C[j]$ has taken all previous pairs into account through the K search trees. The forward propagation makes sure that the search tree queries only consider those pairs where the path endpoint reaches the startpoint of $M[j].P$.

The algorithm is shown in pseudocode in Algorithm 3. There are NK forward propagation links, and both search trees can be queried in $O(\log N)$ time, for a total of $O(NK \log N)$ time. The number of updates on the trees is also bounded by NK, as each endpoint can be contained in at most K

paths. Combined with Theorem 3.2, we have $K = k$ and the total time becomes $O(k|E| \log |V| + kN \log N)$.

Theorem 3.3 *Problem 3.2 on a labeled DAG $G = (V, E, \ell, \Sigma)$ of width k and a set of N input pairs can be solved in time $O(k|E| \log |V| + kN \log N)$ time.*

We now consider how to extend Algorithms 1 and 3 to work for the more general case of Problem 3.1.

We use an FM-index [19] tailored for large alphabets [30] and a two-dimensional range search tree [16] modified to support range maximum queries. The two-dimensional range search trees support range maximum queries in time $O(\log^2 |V|)$. We can show that $O(L)$ queries are sufficient to take all the overlaps into account, where $L = \sum_i |M[i].P|$—the sum of the path lengths—is at most the total input length.

Alternatively, we can process each overlapping pair separately, and compute in constant time its contribution to $C[j]$. This gives another bound $O(L \log \log |V| + \#\texttt{overlaps})$, where $\#\texttt{overlaps}$ is the number of overlaps between the input paths. This can be improved to $O(L + \#\texttt{overlaps})$ by using a generalized suffix tree to compute the overlaps in advance [67, proof of Theorem 2].

Theorem 3.4 *Let $G = (V, E, \ell, \Sigma)$ be a labeled DAG and let M be a set of N pairs of the form $(P, [c..d])$. The algorithms from Theorems 3.1 and 3.3 can be modified to solve Problem 3.1 with additional time $O(L \log^2 |V|)$ or $O(L + \#\texttt{overlaps})$, where L is at most the input length and $\#\texttt{overlaps}$ is the number of overlaps between the input paths.*

The technical details of the FM-index and the two-dimensional range search trees can be found in Paper V.

3.5 Traphlor

In Paper III we added the subpath constraints concept introduced by Bao *et al.* [5], and fully solved by Rizzi *et al.* [67], to our transcript prediction tool Traph (Paper I). Naturally, as the problem to solve is different, the flow engine was replaced with a new version, but we could use the splicing graph module and the module for splitting the flow into paths mostly as is.

3.5.1 Modifications to the splicing graph creation

We used the splicing graph creation module from Traph as the base of the module for Traphlor. To find the subpath constraints we enumerate over the read alignments; every alignment with gapped alignment is added as a potential subpath constraint.

However this process gives all the possible subpath constraints, and as Rizzi *et al.* point out, for the reduction to work correctly any overlapping constraints must first be merged [67]. We do this in two stages. First, we remove all the constraints that are fully contained in another. Second, we merge overlapping constraints.

Rizzi *et al.* proved that iteratively merging the constraints with the longest suffix-prefix overlap is optimal in theory. However, we tested a different approach and found it to perform better in practice.

So instead of merging the constraints with the longest suffix-prefix overlap, we solve the problem using a minimum-cost flow. We use an approach similar to [62]: we create a flow network with the subpath constraints as vertices, and an edge is added between two vertices if the corresponding constraints have a suffix-prefix overlap (see Figure 3.7). The weights on the edges are set based on the difference of coverage between its endpoints. Each edge exiting from the global source of the network is given a weight that is larger than the sum of all the weights in the network, to prefer solutions with the minimum number of paths. The resulting minimum-cost flow is then split into paths: all vertices which belong to the same path are the constraints to be merged.

One thing to note about this flow network based on the constraints (constraint flow network) is that every constraint, that has a flow source of the original graph as its first vertex, is a flow source in the contraint flow network, and every constraint, that has a flow sink of the original graph as its last vertex, is a flow sink in the constraint flow network. Additionally, like in the original graph, every vertex that has no in-neighbors is a flow source, and every vertex that has no out-neighbors is a flow sink. Note that the constraint flow network is not necessarily connected, and a single vertex can be both a flow source and a flow sink if the corresponding constraint does not share any suffix-prefix overlap with other constraints.

After these steps there are no containments or suffix-prefix overlaps between the subpath constraints, and we can proceed to creating the flow network from the splicing graph and the subpath constraints.

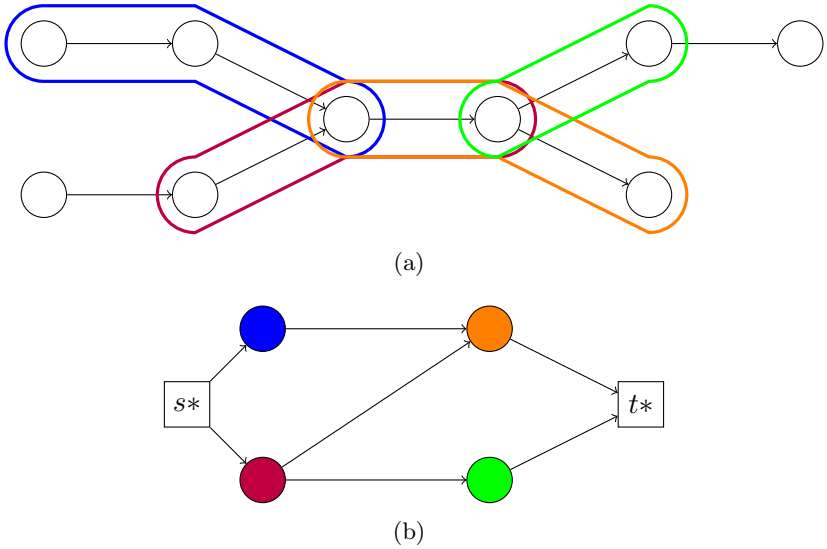


Figure 3.7: a) An example of a splicing graph with subpath constraints and b) the constraint flow network created from them. Square vertices are the global source and sink. Image adapted from Paper III.

3.5.2 The flow engine

As explained in Section 2.3.2 for Traph, the flow network is built from the splicing graph by splitting every vertex v into two vertices v_{in} and v_{out}, adding a global source and sink and connecting the sources and sinks of the splicing graph to them. Adding a high weight (larger than the sum of all weights in the network) on the edges from the global source to splicing graph sources guarantees a solution with the minimum number of paths.

A modification for the subpath constraints goes as follows: for every constraint with first vertex u and last vertex w we add an edge (u, w), with weight equal to the sum of weights on the edges of the subpath. We set the demand of edge (u, w) to 1, and set the demands of the edges that the subpath constraint covers to 0 (see Figure 3.8). That is, instead of covering the edges corresponding to the path in the original network, we have to cover the subpath constraint edge instead.

Given the flow in the graph we can split it into paths greedily like we did with Traph, by repeatedly removing the flow of the path with maximum bottleneck from the network. The subpath edges can be converted back into the corresponding paths trivially. Note that because of the subpath constraint edges, currently Traphlor does not support computing the expression levels like Traph does.

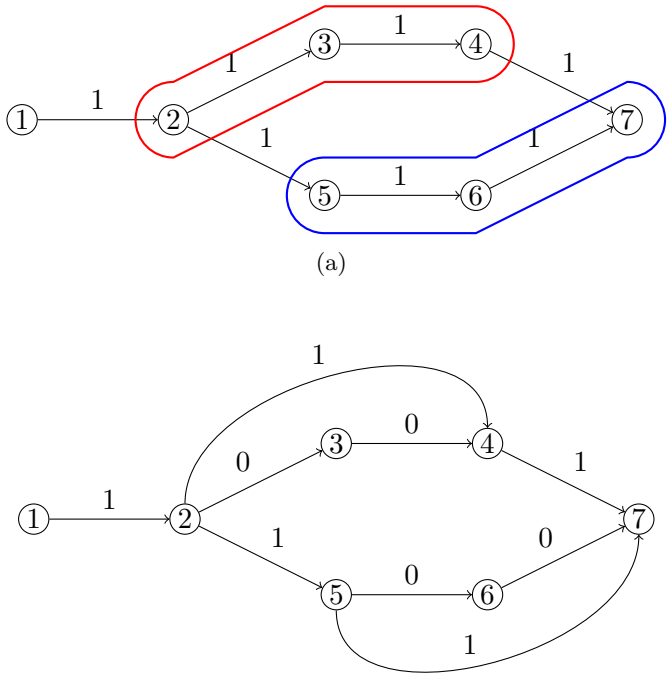
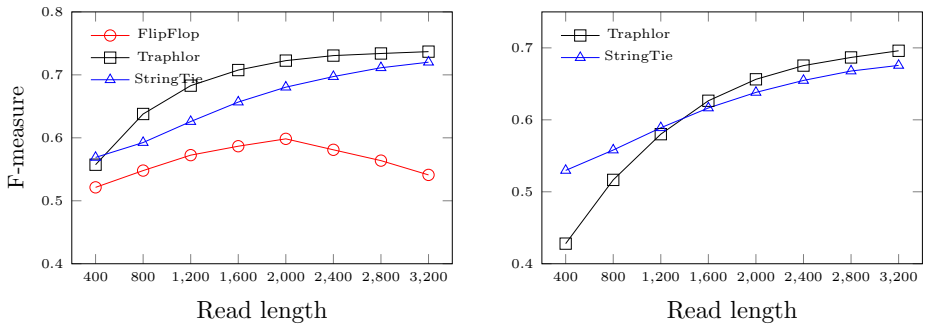


Figure 3.8: Subpath constraints (red and blue in (a)) are converted into edges from the first vertex of the constraint to the last vertex (in (b)). The demands, shown on the graph, on those edges covered by the subpath constraint are set to 0.



(a) Perfect mappings

(b) GMAP mappings

Figure 3.9: F-measure with perfect alignments (a) and GMAP alignments (b). Figure reproduced from Paper III.

3.5.3 Experimental results

We compared Traphlor to two other flow-based transcript assembly tools StringTie [64] and FlipFlop [9]. We tested on simulated data, considering two cases: "perfect alignments" and alignments produced by GMAP [93]. In the latter case FlipFlop required over 50 GB of RAM, and as such we could not run it on the machines available.

Again we chose all the genes in human chromosome 2, but this time with the minimum length of 1 kb. We simulated reads of lengths 400-3,200 bp. Our preliminary tests showed that with realistic long read error profiles (10-15% error rate) GMAP failed to align a significant portion of the reads, so we opted to not add sequencing error to the simulation. To keep the coverage constant we adjusted the number of reads, with there being 60 million 400 bp reads and 7.5 million 3,200 bp reads. Any transcripts shorter than the read length were added to the data set in full.

As Traphlor does not predict expression levels, we only compared the predicted transcripts. Instead of using sequence dissimilarity metrics, as used in Paper I and Paper II, we followed the example of [47] and compared the inner exon borders of the annotated transcripts and predicted transcripts.

As can be seen in Figure 3.9, with perfect mappings Traphlor outperformed its competitors (as measured by F-measure) once read length exceeded 400 bp, but when mapping errors were introduced, the performance dropped significantly. However, Traphlor's performance was on-par with StringTie when read length exceeded 1200 bp. The drop in performance was mainly due to lowered precision, sensitivity was affected only slightly

(see Figures 4 and 5 in Paper III). As we focused on high sensitivity during the development of Traphlor, the drop in precision is an understandable trade-off.

Input: DAG $G = (V, E)$, a path cover $P_1, P_2, \ldots, P_K$ of G, and N pairs $M[1], M[2], \ldots, M[N]$ of the form $(P, [c..d])$.
Output: The index j giving $\max_j C[j]$.
Use Algorithm 2 to find all forward propagation links;
for $i \leftarrow 1$ *to* K **do**
  Initialize search trees $\mathcal{T}_i$ and $\mathcal{I}_i$ with keys $M[j].d$, $1 \leq j \leq N$, and with key 0, all keys associated with values $-\infty$;
  $\mathcal{T}_i$.update$(0, 0)$;
  $\mathcal{I}_i$.update$(0, 0)$;
end
/* Save to start$[i]$ (respectively, end$[i]$) the indexes of all pairs whose path starts (respectively, ends) at i. */
for $j \leftarrow 1$ *to* N **do**
  start$[M[j].P.first]$.push(j);
  end$[M[j].P.last]$.push(j);
end
for $v \in V$ *in topological order* **do**
  for $j \in$ end$[v]$ **do**
    /* Update the search trees for every path that covers v, stored in paths$[v]$. */
    for $i \in$ paths$[v]$ **do**
      $\mathcal{T}_i$.update$(M[j].d, C[j])$;
      $\mathcal{I}_i$.update$(M[j].d, C[j] - M[j].d)$;
    end
  end
  for $(w, i) \in$ forward$[v]$ **do**
    for $j \in$ start$[w]$ **do**
      $C^{\mathsf{a}}[j] \leftarrow (M[j].d - M[j].c + 1) + \mathcal{T}_i$.RMaxQ$(0, M[j].c - 1)$;
      $C^{\mathsf{b}}[j] \leftarrow M[j].d + \mathcal{I}_i$.RMaxQ$(M[j].c, M[j].d)$;
      $C[j] \leftarrow \max(C[j], C^{\mathsf{a}}[j], C^{\mathsf{b}}[j])$;
    end
  end
end
return $\operatorname{argmax}_j C[j]$;

Algorithm 3: Co-linear chaining between a sequence and a DAG using a path cover.

Chapter 4

Conclusions

In this thesis we have discussed RNA-seq data analysis for both short (second-generation sequencing) and long (third-generation sequencing or synthetic) reads. Our main focus has been how to use the information long reads provide about non-consecutive exons in transcript assembly.

First, in the field of short-read analysis, we proposed a novel approach of using minimum-cost flows to solve the transcript assembly problem. We can find the optimal flow in a modified splicing graph, and split the flow into paths using the path of maximum bottleneck decomposition, in polynomial time. We implemented this approach in our tool Traph. We also proposed a version of Traph where instead of looking for a solution with the minimum cost over an unbounded number of paths, we look for exactly k paths with the minimum cost over all such solutions. This is relevant in practice since a small fraction of the graph can be erraneous due to various biological events or technical errors.

We showed that Traph is competitive on transcript prediction accuracy with at-that-time state-of-the-art tools Cufflinks [83], IsoLasso [47] and SLIDE [44].

Compared to exhaustively enumerating all possible candidate paths and then solving a quadratic/integer linear program to evaluate the fitness of each candidate transcript (e.g. IsoInfer/IsoLasso, SLIDE, CLIIQ [48]), minimum-cost flows provide a simple and fast polynomial time algorithm. Cufflinks' minimum path cover approach is done by computing a maximum matching, which can also be reduced to a flow problem. Parallel to our work, a similar minimum-cost flow approach called FlipFlop [9] was proposed. Two years later network flows were used in StringTie [64], which was shown to outperform the state-of-the-art tools, including our Traph.

As part of Traph, we implemented a splicing graph creation module that scales reasonably well to both large read lengths and error rates (as

shown in the running times in Paper V); StringTie, which was chosen for the comparison, had its running time increase exponentially as the read length increased (and was indeed unable to be run in a reasonable time for read lengths exceeding 1200 bp), whereas the running time increase of our module was linear. As the splicing graph creation module used in the experiments of Paper V, SpliceGrapher [69], stalled with highly erraneous long reads, one of the possible directions of future work is converting our graph creation module described in Section 2.3.1 into a stand-alone tool.

After Bao *et al.* [5] proposed, and Rizzi *et al.* [67] fully solved, using long reads as *subpath constraints* in the Minimum Path Cover problem, we extended Traph to use the subpath constraint information. The subpath constraints were modeled as edges in the flow network that spanned from the vertex corresponding to the first exon of the constraint to the vertex corresponding to the last exon of the constraint. Now instead of searching for the solution with the minimum cost, under some cost model, we searched for the minimum number of paths.

For comparison with our tool, Traphlor (for Traph with Long Reads), we chose StringTie and FlipFlop, as their authors had hinted that the tools are capable of using long read information. In the case of "perfect mapping", a situation where all the reads were perfectly mapped to their origin, Traphlor outperformed both its competitors. However, when mapping errors were introduced, precision (the number of matched transcripts divided by the number of predicted transcripts) of Traphlor dropped significantly, and its total accuracy (as measured by F-measure) only reached that of StringTie for read lengths above 1,200 bp. FlipFlop would have required over 50 GB of RAM to execute, and our machines could not provide that.

The loss of precision was most likely due to the fact that Traphlor requires covering all the subpath constraints with some path. This ties back to the problem of long read alignment, where very short segments after a splice junction can remain unmapped in the seeding stage. Most one-pass mapping algorithms (including GMAP [93], which we used) prefer extending the alignment to include the first few bases of the intervening intron instead of the correct mapping. The result was that Traphlor reported a transcript for each of these mismappings.

As the mappings will likely include even more of these short anchors when sequencing errors are added to the picture, an important direction for future work on Traphlor is making the algorithm more resistant to noise. Whether an expression level quantification step can be added is also an interesting direction of future work, although there exist many tools that handle the quantification step when given the transcript sequences.

Our work with Traphlor brought up an interesting point on what objective to optimize: instead of optimizing the local alignment score for each read, we could optimize the correctness of the exon chains that make up the subpath constraint instead. Our third contribution was evaluating different methods for optimizing the correctness of the exon chains. We considered three approaches: creating a splicing graph from short reads and using dynamic programming to align long reads to the graph, a fast approximation of the above by using overlaps between the short and long read alignments, and error-correcting the long reads using short reads before mapping them to the reference genome. As the base case we created the splicing graph from both short and long reads but, as mentioned, the tool SpliceGrapher could not handle this case in a reasonable time with reads exceeding 400 bp.

Our results showed that error-correcting the reads yielded the best results, but using the coordinate overlaps between short and long reads was not far behind. Dynamic programming performed poorly, possibly due to the same short-anchor issue that Traphlor had encountered. Dynamic programming is guaranteed to find an optimal solution, but it can break ties arbitrarily.

While the dynamic programming solution proved to be ineffective, both accuracy and running time wise, for finding the subpath constraints, we pursued the direction of aligning a sequence to a graph. While it is unlikely there exists an algorithm for aligning sequences in subquadratic time (as it would violate the Strong Exponential Time Hypothesis [4]), co-linear chaining problem between two sequences with N anchor pairs can be solved in $O(N \log N)$ time [2].

Our last contribution was to extend co-linear chaining from between two sequences to between a sequence and a DAG. Recently Patro *et al.* proposed a transcript quantification algorithm that only requires the co-linear chain of the spliced alignment instead of full alignment [63], to which our approach could be applicable.

Our solution is based on covering the DAG with k paths, and treating these paths as sequences. Careful bookkeeping of the processing order is required to make sure all the in-neighbors of the vertex are processed before the vertex itself. This problem formulation is very generic, and can be used for extending other dynamic programming tasks (e.g. longest increasing subsequence or longest common subsequence) from sequences to between a sequence and a DAG.

First we considered a relaxed variant of a problem, where the anchors in the DAG were not allowed to overlap. Given the k paths and N an-

chor pairs, we can solve this relaxed variant in $O(kN \log N)$ time. When the DAG consists of an unary sequence ($k = 1$), the time requirement matches that of the optimal solution for the sequences. For additional time $O(L \log^2 |V|)$ or $O(L + \#\texttt{overlaps})$, where L is at most the input length and $\#\texttt{overlaps}$ is the number of overlaps between the input paths, we can allow for overlaps in the anchors in the DAG.

There are two main directions for future work for co-linear chaining between a sequence and a DAG. First, it remains open whether the bound $O(k|E| \log |V|)$ for the MPC problem can be improved. Second, we assumed that we are given N anchor pairs as an input. With sequences, the anchors are usually taken to be *maximal exact matches* (MEMs) and can be retrieved in linear time [8, 7]. While practical approaches for retrieving length-limited MEMs between a sequence and a DAG exists [77, 76], it is largely an open problem how to retrieve MEMs between a sequence and a DAG in general.

References

[1] Pacific Biosciences. SMRT sequencing: read lengths, http://www.pacb.com/smrt-science/smrt-sequencing/read-lengths/ (May 19, 2017, date last accessed).

[2] M. I. Abouelhoda. A Chaining Algorithm for Mapping cDNA Sequences to Multiple Genomic Sequences. In *14th International Symposium on String Processing and Information Retrieval*, volume 4726 of *LNCS*, pages 1–13. Springer, 2007.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[4] A. Backurs and P. Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58. ACM, 2015.

[5] E. Bao, T. Jiang, and T. Girke. BRANCH: boosting RNA-Seq assemblies with partial or related genomic sequences. *Bioinformatics*, 29(10):1250–1259, 2013.

[6] G. Baruzzo, K. E. Hayer, E. J. Kim, B. Di Camillo, G. A. FitzGerald, and G. R. Grant. Simulation-based comprehensive benchmarking of RNA-seq aligners. *Nature Methods*, 14(2):135–139, 2017.

[7] D. Belazzougui. Linear time construction of compressed text indices in compact space. In *Proc. Symposium on Theory of Computing STOC 2014*, pages 148–193. ACM, 2014.

[8] D. Belazzougui, F. Cunial, J. Kärkkäinen, and V. Mäkinen. Versatile Succinct Representations of the Bidirectional Burrows-Wheeler Transform. In *Proc. 21st Annual European Symposium on Algorithms (ESA 2013)*, volume 8125 of *LNCS*, pages 133–144. Springer, 2013.

[9] E. Bernard, L. Jacob, J. Mairal, and J.-P. Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, Sep 2014.

[10] I. Birol, S. D. Jackman, C. B. Nielsen, J. Q. Qian, R. Varhol, G. Stazyk, R. D. Morin, Y. Zhao, M. Hirst, J. E. Schein, et al. De novo transcriptome assembly with abyss. *Bioinformatics*, 25(21):2872–2877, 2009.

[11] N. Bray, I. Dubchak, and L. Pachter. AVID: A global alignment program. *Genome Research*, 13(1):97–102, 2003.

[12] M. Brudno, S. Malde, A. Poliakov, C. B. Do, O. Couronne, I. Dubchak, and S. Batzoglou. Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(suppl_1):i54–i62, 2003.

[13] M. J. Chaisson and T. Glenn. Mapping single molecule sequencing reads using Basic Local Alignment with Successive Refinement (BLASR): Theory and Application. *BMC Bioinformatics*, 13:238, 2012.

[14] R. Cilibrasi and P. M. Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[15] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno. SHRiMP2: sensitive yet practical SHort Read Mapping. *Bioinformatics*, 27(7):1011–1012, Apr 2011.

[16] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[17] P. G. Engström, T. Steijger, B. Sipos, G. R. Grant, A. Kahles, G. Rätsch, N. Goldman, T. J. Hubbard, J. Harrow, R. Guigó, et al. Systematic evaluation of spliced alignment programs for RNA-seq data. *Nature methods*, 10(12):1185–1191, 2013.

[18] J. Feng, W. Li, and T. Jiang. Inference of isoforms from short sequence reads. In B. B, editor, *RECOMB*, volume 6044 of *Lecture Notes in Computer Science*, pages 138–157. Springer, 2010.

[19] P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

[20] L. Florea, V. Di Francesco, J. Miller, R. Turner, A. Yao, M. Harris, B. Walenz, C. Mobarry, G. V. Merkulov, R. Charlab, I. Dew, Z. Deng,

S. Istrail, P. Li, and G. Sutton. Gene and alternative splicing annotation with AIR. *Genome Res*, 15(1):54–66, Jan 2005.

[21] S. Goodwin, J. D. McPherson, and W. R. McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

[22] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, et al. Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, 29(7):644–652, 2011.

[23] G. R. Grant, M. H. Farkas, A. D. Pizarro, N. F. Lahens, J. Schug, B. P. Brunk, C. J. Stoeckert, J. B. Hogenesch, and E. A. Pierce. Comparative analysis of RNA-Seq alignment algorithms and the RNA-Seq unified mapper (RUM). *Bioinformatics*, 27(18):2518–2528, Sep 2011.

[24] M. Guttman, M. Garber, J. Z. Levin, J. Donaghey, J. Robinson, X. Adiconis, L. Fan, M. J. Koziol, A. Gnirke, C. Nusbaum, et al. Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincrnas. *Nature biotechnology*, 28(5):503–510, 2010.

[25] K. D. Hansen, S. E. Brenner, and S. Dudoit. Biases in Illumina transcriptome sequencing caused by random hexamer priming. *Nucleic Acids Res*, 38(12):e131, Jul 2010.

[26] K. E. Hayer, A. Pizarro, N. F. Lahens, J. B. Hogenesch, and G. R. Grant. Benchmark analysis of algorithms for determining and quantifying full-length mrna splice forms from rna-seq data. *Bioinformatics*, 31(24):3938–3945, 2015.

[27] S. Heber, M. Alekseyev, S.-H. Sze, H. Tang, and P. A. Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, 18 Suppl 1:S181–S188, 2002.

[28] J. Henderson, S. Salzberg, and K. H. Fasman. Finding genes in DNA with a Hidden Markov Model. *J Comput Biol*, 4(2):127–141, 1997.

[29] M. J. Holland. Transcript abundance in yeast varies over six orders of magnitude. *J Biol Chem*, 277(17):14363–14366, Apr 2002.

[30] W. Hon, K. Sadakane, and W. Sung. Breaking a time-and-space barrier in constructing full-text indices. *SIAM J. Comput.*, 38(6):2162–2178, 2009.

[31] S. Huang, J. Zhang, R. Li, W. Zhang, Z. He, T.-W. Lam, Z. Peng, and S.-M. Yiu. SOAPsplice: Genome-Wide ab initio Detection of Splice Junctions from RNA-Seq Data. *Front Genet*, 2:46, 2011.

[32] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome, 2001.

[33] M. Jain, H. E. Olsen, B. Paten, and M. Akeson. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1):239, 2016.

[34] H. Jiang and W. H. Wong. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20):2395–2396, 2008.

[35] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome biology*, 14(4):R36, 2013.

[36] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden Markov model for the recognition of human genes in DNA. *Proc Int Conf Intell Syst Mol Biol*, 4:134–142, 1996.

[37] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

[38] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[39] Lemon. **L**ibrary for **E**fficient **M**odeling and **O**ptimization in **N**etworks. http://lemon.cs.elte.hu/, 2014.

[40] H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv:1303.3997v1, 2013.

[41] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[42] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[43] H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, 18(11):1851–1858, 2008.

[44] J. J. Li, C.-R. Jiang, J. B. Brown, H. Huang, and P. J. Bickel. Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proc. Natl. Acad. Sci. U. S. A.*, 108(50):19867–19872, Dec 2011.

[45] R. Li, Y. Li, K. Kristiansen, and J. Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[46] W. Li. http://alumni.cs.ucr.edu/˜liw/rnaseqreadsimulator.html, 2012.

[47] W. Li, J. Feng, and T. Jiang. IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. *J Comput Biol*, 18(11):1693–1707, Nov 2011.

[48] Y.-Y. Lin, P. Dao, F. Hach, M. Bakhshi, F. Mo, A. Lapuk, C. Collins, and S. Sahinalp. CLIIQ: Accurate Comparative Detection and Quantification of Expressed Isoforms in a Population. *Proc Algorithms in Bioinformatics - 12th International Workshop, WABI 2012*, Volume 7534 of Lecture Notes in Computer Science:178–189, 2012.

[49] R. Lindner and C. C. Friedel. A comprehensive evaluation of alignment algorithms in the context of RNA-seq. *PLoS One*, 7(12):e52403, 2012.

[50] J. Liu, T. Yu, T. Jiang, and G. Li. TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs. *Genome biology*, 17(1):213, 2016.

[51] Y. Liu, B. Popp, and B. Schmidt. CUSHAW3: sensitive and accurate base-space and color-space short-read alignment with hybrid seeding. *PloS one*, 9(1):e86869, 2014.

[52] P. C. Lott and I. Korf. StochHMM: a flexible hidden Markov model tool and C++ library. *Bioinformatics*, 30(11):1625–1626, 2014.

[53] R. Lowe, N. Shirley, M. Bleackley, S. Dolan, and T. Shafee. Transcriptomics technologies. *PLoS computational biology*, 13(5):e1005457, 2017.

[54] A. V. Lukashin and M. Borodovsky. GeneMark.hmm: new solutions for gene finding. *Nucleic Acids Res*, 26(4):1107–1115, Feb 1998.

[55] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[56] V. Mäkinen, D. Belazzougui, F. Cunial, and A. I. Tomescu. *Genome-Scale Algorithm Design—Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, May 2015.

[57] V. Mäkinen, L. Salmela, and J. Ylinen. Normalized N50 assembly metric using gap-restricted co-linear chaining. *BMC Bioinformatics*, 13:255, 2012.

[58] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca. The GEM mapper: fast, accurate and versatile alignment by filtration. *Nature methods*, 9(12):1185–1188, 2012.

[59] L. Maretty, J. A. Sibbesen, and A. Krogh. Bayesian transcriptome assembly. *Genome biology*, 15(10):501, 2014.

[60] J. A. Martin and Z. Wang. Next-generation transcriptome assembly. *Nat Rev Genet*, 12(10):671–682, Oct 2011.

[61] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

[62] S. C. Ntafos and S. L. Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, SE-5(5):520–529, 1979.

[63] R. Patro, G. Duggal, M. I. Love, R. A. Irizarry, and C. Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 04 2017.

[64] M. Pertea, G. M. Pertea, C. M. Antonescu, T.-C. Chang, J. T. Mendell, and S. L. Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nat Biotechnol*, 33(3):290–295, Mar 2015.

[65] F. Rapaport, R. Khanin, Y. Liang, M. Pirun, A. Krek, P. Zumbo, C. E. Mason, N. D. Socci, and D. Betel. Comprehensive evaluation of differential gene expression analysis methods for RNA-seq data. *Genome Biol*, 14(9):R95, Sep 2013.

[66] A. Rhoads and K. F. Au. PacBio sequencing and its applications. *Genomics, proteomics & bioinformatics*, 13(5):278–289, 2015.

[67] R. Rizzi, A. I. Tomescu, and V. Mäkinen. On the complexity of Minimum Path Cover with Subpath Constraints for multi-assembly. *BMC Bioinformatics*, 15(S-9):S5, 2014.

[68] J. T. Robinson, H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, 2011.

[69] M. F. Rogers, J. Thomas, A. S. Reddy, and A. Ben-Hur. SpliceGrapher: detecting patterns of alternative splicing from RNA-Seq data in the context of gene models and EST data. *Genome biology*, 13(1):R4, 2012.

[70] L. Salmela and E. Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, page btu538, 2014.

[71] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977.

[72] E. E. Schadt, S. Turner, and A. Kasarskis. A window into third-generation sequencing. *Human molecular genetics*, 19(R2):R227–R240, 2010.

[73] M. H. Schulz, D. R. Zerbino, M. Vingron, and E. Birney. Oases: robust de novo rna-seq assembly across the dynamic range of expression levels. *Bioinformatics*, 28(8):1086–1092, 2012.

[74] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human–mouse alignments with BLASTZ. *Genome research*, 13(1):103–107, 2003.

[75] D. Sharon, H. Tilgner, F. Grubert, and M. Snyder. A single-molecule long-read survey of the human transcriptome. *Nat Biotechnol*, 31(11):1009–1014, Nov 2013.

[76] J. Sirén. Indexing Variation Graphs. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 13–27. SIAM, 2017.

[77] J. Sirén, N. Välimäki, and V. Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

[78] L. Song and L. Florea. CLASS: constrained transcript assembly of RNA-seq reads. *BMC Bioinformatics*, 14 Suppl 5:S14, 2013.

[79] M. Stanke and S. Waack. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*, 19(suppl 2):ii215–ii225, 2003.

[80] S. Thomas, J. G. Underwood, E. Tseng, A. K. Holloway, et al. Long-read sequencing of chicken transcripts and identification of new transcript isoforms. *PloS one*, 9(4):e94650, 2014.

[81] H. Tilgner, F. Jahanbani, T. Blauwkamp, A. Moshrefi, E. Jaeger, F. Chen, I. Harel, C. D. Bustamante, M. Rasmussen, and M. P. Snyder. Comprehensive transcriptome analysis using synthetic long-read sequencing reveals molecular co-association of distant splicing events. *Nature biotechnology*, 33(7):736–742, 2015.

[82] C. Trapnell, L. Pachter, and S. L. Salzberg. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, May 2009.

[83] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. J. van Baren, S. L. Salzberg, B. J. Wold, and L. Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat Biotechnol*, 28(5):511–515, May 2010.

[84] R. Uricaru, C. Michotey, H. Chiapello, and E. Rivals. YOC, A new strategy for pairwise alignment of collinear genomes. *BMC Bioinformatics*, 16(1):111, Apr 2015.

[85] B. Vatinlen, F. Chauvet, P. Chrétienne, and P. Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008.

[86] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[87] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[88] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt. A Long Fragment Aligner called ALFALFA. *BMC Bioinformatics*, 16(1):159, May 2015.

[89] M. Vyverman, D. De Smedt, Y.-C. Lin, L. Sterck, B. De Baets, V. Fack, and P. Dawyndt. Fast and Accurate cDNA Mapping and Splice Site Identification. In *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms (BIOSTEC 2014)*, pages 233–238, 2014.

[90] S. Wandelt and U. Leser. RRCA: ultra-fast multiple in-species genome alignments. In *Algorithms for Computational Biology - First International Conference, AlCoB 2014, Tarragona, Spain, July 1-3, 2014, Proceedigns*, pages 247–261, 2014.

[91] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou, J. N. MacLeod, D. Y. Chiang, J. F. Prins, and J. Liu. MapSplice: accurate mapping of RNA-seq reads for splice junction discovery. *Nucleic Acids Res*, 38(18):e178, Oct 2010.

[92] Z. Wang, M. Gerstein, and M. Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet*, 10(1):57–63, Jan 2009.

[93] T. D. Wu and C. K. Watanabe. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics*, 21(9):1859–1875, May 2005.

[94] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.