OXFORD

*Briefings in Bioinformatics*, 19(3), 2018, 404–414

**doi: 10.1093/bib/bbw137**
Advance Access Publication Date: 9 January 2017
Paper

# Evaluating approaches to find exon chains based on long reads

Anna Kuosmanen, Tuukka Norri and Veli Mäkinen

Corresponding author. Veli Mäkinen, Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, Helsinki, Finland. Tel.: +3582941911; E-mail: veli.makinen@helsinki.fi

## Abstract

Transcript prediction can be modeled as a graph problem where exons are modeled as nodes and reads spanning two or more exons are modeled as exon chains. Pacific Biosciences third-generation sequencing technology produces significantly longer reads than earlier second-generation sequencing technologies, which gives valuable information about longer exon chains in a graph. However, with the high error rates of third-generation sequencing, aligning long reads correctly around the splice sites is a challenging task. Incorrect alignments lead to spurious nodes and arcs in the graph, which in turn lead to incorrect transcript predictions. We survey several approaches to find the exon chains corresponding to long reads in a splicing graph, and experimentally study the performance of these methods using simulated data to allow for sensitivity/precision analysis. Our experiments show that short reads from second-generation sequencing can be used to significantly improve exon chain correctness either by error-correcting the long reads before splicing graph creation, or by using them to create a splicing graph on which the long-read alignments are then projected. We also study the memory and time consumption of various modules, and show that accurate exon chains lead to significantly increased transcript prediction accuracy. Availability: The simulated data and in-house scripts used for this article are available at http://www.cs.helsinki.fi/group/gsa/exon-chains/exon-chains-bib.tar.bz2.

**Key words:** alternative splicing; transcript prediction; RNA sequencing; split-read alignment

## Introduction

Third-generation sequencers Pacific Biosciences (PacBio) and Oxford Nanopore increased the sequencing read length tremendously compared with the next-generation platform Illumina. Whereas Illumina read lengths vary from ∼75 bases to 400 bases, PacBio platform generates reads up to several tens of thousands bases long.

However, because of library preparation limitations, full-length reads for transcripts longer than 2.5 kilobases are less likely to be sequenced [1]. But even non-full-length long reads can give valuable information about nonadjacent exons in a transcript: Assuming a genome reference is available, one can ‘align’ the long reads allowing introns to be spliced out in the alignment. Then one can read an ‘exon chain’ corresponding to the read alignment, that is the sequence of exons that the read overlaps.

The prediction of full transcripts can be modeled as a combinatorial problem in a ‘splicing graph’ [2], where nodes are exons and arcs are exons consecutive in some read alignment. Rizzi *et al.* [3] proposed modeling long reads as subpath constraints (exon chains) in a splicing graph. Figure 1 illustrates these concepts.

**Anna Kuosmanen** is a PhD student at the Department of Computer Science of the University of Helsinki. Her research interests lie in developing methods for analysis of third-generation RNA sequencing reads.
**Tuukka Norri** is a Research Assistant at the Department of Computer Science of the University of Helsinki. He is starting his doctoral studies with focus on generalizations of the Burrows-Wheeler transform.
**Veli Mäkinen** is a Professor at the Department of Computer Science of the University of Helsinki. His research focuses on algorithmic approaches to genomic data analysis. He is a coauthor of a textbook on Genome-Scale Algorithm Design.
**Submitted:** 19 August 2016; **Received (in revised form):** 1 December 2016

Kuosmanen *et al.* [4] implemented the concepts proposed by Rizzi *et al.* in the transcript prediction tool Traphlor, and using simulated data demonstrated against two state-of-the-art transcript prediction tools StringTie [5] and FlipFlop [6] that using the information provided by exon chains can significantly improve the transcript prediction accuracy.

However, with the high error rates of the third-generation sequencers (~15% for PacBio and up to 45% for Nanopore, compared with the 1% error rate of second-generation sequencers), aligning the long reads correctly around the splice sites is a difficult task (however, see PacBio 'Reads Of Insert' for a potential technique to lower down the error rate). An additional challenge is posed by the error types: second-generation sequencing errors are generally substitutions, whereas third-generation sequencing errors are mostly insertions and deletions. Incorrect alignments lead to spurious extra nodes and arcs in the splicing graph, which in turn lead to erroneous transcript predictions.

In Figures 2 and 3, we have replicated the experiments of Kuosmanen *et al.* on a smaller data set simulated from human chromosome 2. As can be seen in the figures, introducing even just mapping errors (as a reminder, in the experimental setup of [4], no sequencing errors were simulated) causes the accuracy of the prediction to decrease significantly.

As suggested by Kuosmanen *et al.* [4], one approach to tackle the problem of alignment errors near the splice sites would be to optimize the correctness of the chain of exons instead of the alignment of each long read separately.

In this article, we survey several approaches to find the exon chains that correspond to the long reads in a splicing graph, as well as provide an experimental study on the performance of these methods. Many of the techniques surveyed are well known in the literature, but their combination and experimental evaluation toward the identification of exon chains have not been carried out previously. Our study gives several new insights into the feasibility of the problem, and proposes important directions for further studies (see Discussion section).

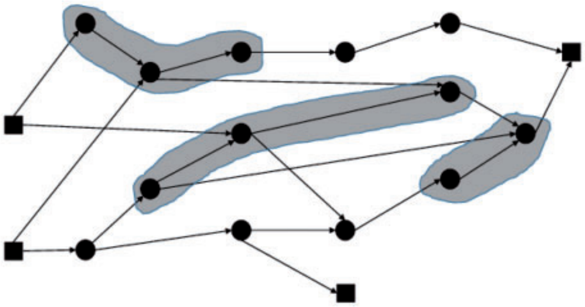


**Figure 1.** An example of a splicing graph, in which three exon chains are drawn in gray. The square nodes are the ones where the transcripts can start or end. Exon chains limit the way transcripts could be formed, as each chain should be contained in at least in one transcript.

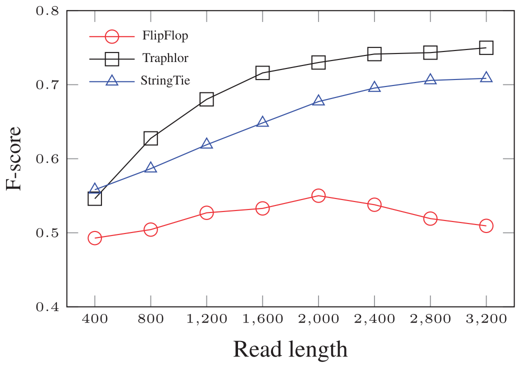


**Figure 2.** *F*-score, the harmonic mean of sensitivity and precision, of different transcript prediction tools, StringTie [5], FlipFlop [6] and Traphlor [4], using 'perfect alignments'; alignment information was gathered directly from simulated reads mimicing the use of a perfect aligner.

## Methods

In this section, we introduce four methods that can be used for finding exon chains; one of them can also be used as a preprocessor for the other three. As the input, we assume there to be both 'short reads' and 'long reads' from the RNA transcripts. In addition, we assume the reference genome of the species under study to be available, so that we can exploit RNA to DNA read alignments. We consider short reads to be reads of lengths 75–250 with error profile consisting mostly of substitutions, which is typical to the most commonly used second-generation sequencing platform Illumina. Long reads, on the other hand, are reads with lengths from several hundred to several thousand bases, produced by third-generation sequencing platforms such as PacBio and Oxford Nanopore, and their dominant error type is insertions, followed by deletions. The error rate on the long reads is also a magnitude higher than on the short reads (15–45 versus 1%).

The first step in finding exon chains is to build 'a splicing graph' [2]. In a splicing graph, nodes correspond to exons, and there is an edge between two nodes $v_i$ and $v_j$ if there exists a split-read alignment where the exons corresponding to nodes $v_i$ and $v_j$ are consecutive. This is illustrated in Figure 4.

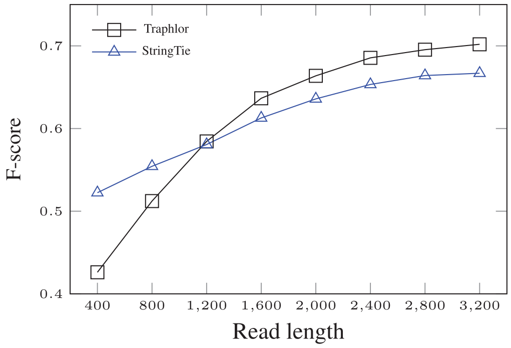


**Figure 3.** *F*-score of different transcript prediction tools using GMAP software [7] for aligning error-free reads.

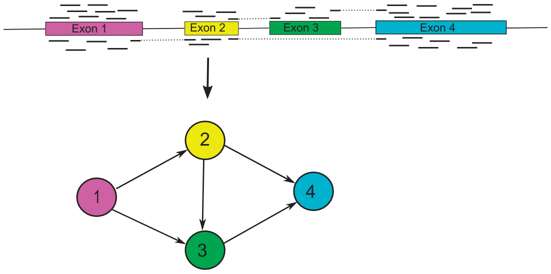


**Figure 4.** In a splicing graph, exons are represented by nodes, and there is an arc between two nodes if the corresponding exons are consecutive in some read alignment.

Our baseline method is to build the splicing graph using both the short and long reads, and find the exon chains corresponding to the long reads in the resulting splicing graph by matching the genomic coordinates (see ‘Merged alignments: creating splicing graph from both short and long reads’ section for details). As accurate long-read alignment around the splice sites is difficult (the very fact that prompted this study), using the long reads as is alone is not feasible.

For comparison, we consider aligning the long reads directly on a splicing graph created from the short reads using dynamic programming (see ‘Dynamic programming: aligning long reads to a splicing graph’ section for details), as well as a more approximate, but significantly faster, approach of considering the overlaps in genomic coordinates to infer the exon chains (see ‘Overlaps: inferring paths in splicing graph from overlaps between exons and aligned long reads’ section for details).

Additionally, we talk about error-correcting the long reads with short reads (see ‘Error-correction: correcting long reads with short reads’ section for details), which can be used both as a preprocessing step as well as a stand-alone approach. As error-correction method significantly increases the mappability of the reads, the alignments around the splice sites are more reliable than when using raw long reads, and the long-read alignments alone can be used for inferring the splicing graph.

## Error-correction: correcting long reads with short reads

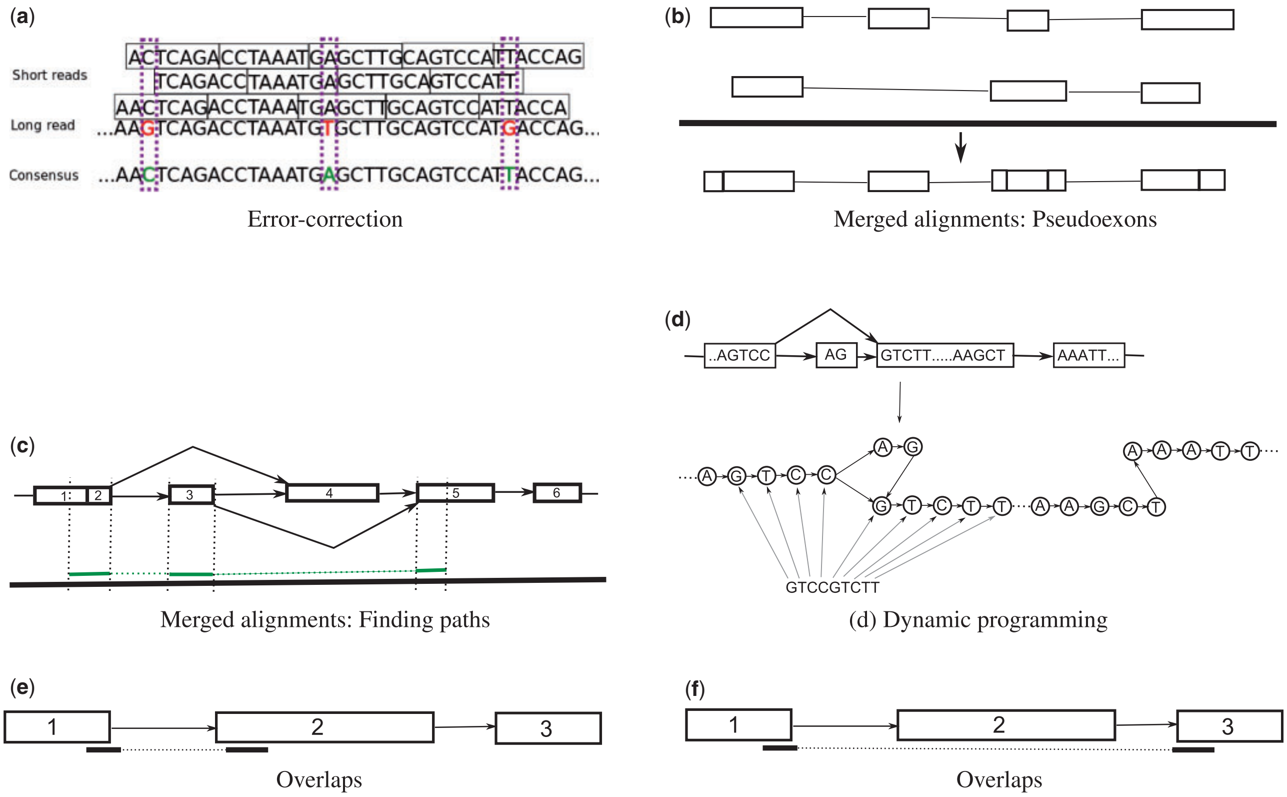
Although PacBio reads have high error rate (> 15%) [8], and as such pose a harder challenge to error-correction than next-generation sequencing reads, the errors seem to be uniformly distributed and independent of the sequence context. Because of this type of error profile, consensus-based methods are suitable for the problem.

There are two main approaches for error-correcting long reads: ‘self-correction’ and ‘hybrid correction’. Self-correction uses only long reads, and creates a consensus sequence by computing local alignments between the long reads. Hybrid correction uses more accurate short next-generation sequencing reads to create a consensus by aligning them on the long reads (Figure 5A).

For this study, we use error-correction tool LoRDEC [9]. LoRDEC is a hybrid correction method that first builds a de Brujin graph (DBG) from short reads, and then uses the DBG to correct erroneous regions within each long read individually. It has been shown to make most of the sequence alignable with percentage of identity >97%, and to do so with significantly lower running time than any previous self-correction tools.

## Merged alignments: creating splicing graph from both short and long reads

Our baseline method for this study is to align both short and long reads to the reference genome, and use both of these


(a)
Short reads
ACTCAGACCTAAATGAGCTTGCAGTCCATTACCAG
TCAGACCTAAATGAGCTTGCAGTCCATT
AACTCAGACCTAAATGAGCTTGCAGTCCATTACCA
Long read
...AAGTCAGACCTAAATGTGCTTGCAGTCCATGACCAG...
Consensus
...AACTCAGACCTAAATGAGCTTGCAGTCCATTACCAG...
(b)
(c)
1
2
3
4
5
6
(d)
..AGTCC
AG
GTCTT.....AAGCT
AAATT...
A G T C C
A G
G T C T T A A G C T
A A A T T
GTCCGTCTT
(e)
1
2
3
(f)
1
2
3

Error-correction

Merged alignments: Pseudoexons

Merged alignments: Finding paths

(d) Dynamic programming

Overlaps

Overlaps

Figure 5. Four approaches to improve exon chain detection: error-correction (a), merged alignments (b and c), dynamic programming (d) and overlaps (e and f): (A) Long reads with high error rate can be corrected by aligning more accurate short reads on them and taking the consensus of the column. In this example, the errors in the long read are marked on red, and the corrected bases are marked on green in the consensus sequence. (B) In transcripts, two exons can have overlapping genomic coordinates (top). We split the overlapping exons into nonoverlapping ‘pseudoexons’ (bottom). (C) The path corresponding to the long-read alignment (green) can be read by comparing the genomic coordinates of the long-read alignment and the pseudoexons. In this case, the corresponding path is ‘1, 2, 3, 5’. (D) From a splicing graph, we create a sequence graph where each base in the sequence is a node. A read sequence can then be aligned to this base graph using dynamic programming. (E) The read overlaps nodes 1 and 2 in genomic coordinates, resulting to potential path of ‘1 2’. As there is an arc between nodes 1 and 2, the path is accepted. (F) The read overlaps nodes 1 and 3, resulting to potential path of ‘1 3’. However, there is no arc between the nodes 1 and 3 in the splicing graph, so the path is rejected.

alignment sets to infer the splicing graph. As mentioned earlier, in a splicing graph, exons are represented by nodes, and there is an arc between two nodes if the corresponding exons are consecutive in some read alignment.

The naive approach on locating the exons is to examine the read coverage at every genomic position: positions where the coverage $c > 0$ are designated exonic, and areas where $c = 0$ are designated intronic. Additional information about the exon borders contained within another exon can be found from split-read alignments.

Instead of this naive approach, one can use more sophisticated methods for splicing graph creation, using, for example, software tool SpliceGrapher [10].

SpliceGrapher takes as input the read alignment file and gene model annotation that will be used as the base of the graph. From the gene model, SpliceGrapher adds all the exon entries as nodes, and sets arcs between them if they are consecutive in some transcript. Then, it constructs what the authors call 'short-read exons' from clusters of contiguous ungapped alignments. Although these are called short-read exons, ungapped regions of long-read alignments are also included. Any short-read exons contained within the exons inferred from the gene model are discarded to avoid duplicates. Acceptor and donor sites are predicted from spliced alignments, and the remaining short-read exons are extended to the nearest splice site if they do not already reach it. The extended short-read exons are then added to the model as novel nodes, and alignments spanning two short-read exons are added as novel arcs.

It is possible that there exist two exons that overlap in genomic coordinates (but do not have the exactly same start and end coordinates, which would make them the same exon). As a post-processing step for the splicing graph creation, we split any overlapping exons into 'pseudoexons' as illustrated in Figure 5B. Unlike exons, pseudoexons cannot have any overlap between them, which simplifies the next steps.

The paths corresponding to the long-read alignments can be read from the graph by comparing the start and end coordinates of the blocks (i.e. continuous sequences) in the read alignment with the coordinates of the nodes (pseudoexons). That is, for a node to be considered a candidate in the path, either the start and end coordinates for some block in the alignment have to match the start and end coordinates of the node, or the node has to be completely contained in some block in the alignment. For the first and last block of the read alignment, only the end coordinates (respectively, start) have to match the coordinates of the node. See Figure 5C for illustration.

If some block of the alignment did not match any node as described above, the alignment is reported to fail to find a path. Also, if there is no arc in the splicing graph between two candidate nodes $v_i$ and $v_j$ that are adjacent in the path, the alignment is also reported to fail.

Note that as the long-read alignments were also used in the construction of the graph, in theory, there should be a path corresponding to every long-read alignment. However, depending on the implementation details of the splicing graph construction algorithm, some alignments or parts of them may be discarded. Our experiments indicate that such cases occur with SpliceGrapher.

## Dynamic programming: aligning long reads to a splicing graph

We assume a splicing graph has been created using short reads. One can then apply dynamic programming to align a long read into a splicing graph (Section 6.6.5 of [11]). Although this approach is guaranteed to be optimal for the long read in question, other approaches that consider all reads at once may still perform better. (Multiple alignment formulation would give an optimal model, but is infeasible in practice, and will not be discussed here further.)

For self-containedness, we briefly review the approach described in (Section 6.6.5 of [11]). Denote a directed acyclic graph (DAG) $G$ as $(V^G, E^G)$, where $V^G$ is the set of vertices, and $E^G$ is the set of arcs. A 'labeled DAG' has a label $\ell(v)$ for each vertex $v \in V^G$.

'DAG-path alignment problem' is defined as follows. Given two labeled DAGs, $A = (V^A, E^A)$ and $B = (V^B, E^B)$, both with a unique source and sink, find a source-to-sink path $P^A$ in $A$, and a source-to-sink path $P^B$ in $B$, such that the global alignment score $S(\ell(P^A), \ell(P^B))$ is maximum over all such pairs of paths. Here, notion $\ell()$ returns the concatenation of vertex labels on the path given as argument.

This problem is easy to solve by extending the global alignment dynamic programming formulation. Consider the recurrence:

$$s_{i,j} = \max \begin{cases} \max_{i' \in N_A^-(i), j' \in N_B^-(j)} s_{i',j'} + s(\ell^A(i), \ell^B(j)), \\ \max_{i' \in N_A^-(i)} s_{i',j} - \delta, \\ \max_{j' \in N_B^-(j)} s_{i,j'} - \delta, \end{cases},$$

where $i$ and $j$ denote the vertices of $A$ and $B$ with their topological ordering number, respectively, $N_A^-(\cdot)$ and $N_B^-(\cdot)$ are the functions giving the set of in-neighbors of a given vertex and $\ell^A(\cdot)$ and $\ell_B(\cdot)$ are the functions giving the corresponding vertex labels. Initializing $s_{0,0} = 0$ and evaluating the values in a suitable evaluation order (e.g. $i = 1$ to $|V^A|$ and $j = 1$ to $|V^B|$), one can see that $s_{|V^A|,|V^B|}$ evaluates to the solution of the DAG-path alignment: an optimal alignment can be extracted with a traceback. The algorithm takes $O(|E^A||E^B|)$ time (Section 6.6.5 of [11]).

One can now set $A$ as a linear labeled DAG denoting the long read and set $B$ as the sequence graph corresponding to the splicing graph (Figure 5D). DAG-path alignment on these inputs gives the best alignment of the long read to the splicing graph. To allow a long read to match a subpath (exon chain) instead of a full transcript, one needs to modify the approach for 'semi-local alignment'. This is accomplished by initializing $s_{0,j} = 0$ for all $j$. After the dynamic programming, one can look for node $j$ with the maximum value $s_{m,j}$, where $m$ is the length of the long read. Traceback from this node reveals an exon chain containing a best match to the read.

For the scores for the dynamic programming formula, we used the following: match = 4, mismatch =−3, insertion =−2 and deletion =−2. This scheme slightly favors insertions and deletions over mismatches, as is appropriate for long-read error profile. Choosing lower penalties for insertion and deletion (or higher for substitution) is not possible, as in that case, it would be better to insert and delete instead of reporting a mismatch.

## Overlaps: inferring paths in splicing graph from overlaps between exons and aligned long reads

Next, we consider a hybrid approach, where we again assume a splicing graph is created using short reads only. But in this case, the long reads are aligned to the reference instead of aligning them to the splicing graph. The long-read alignments can then be projected to the splicing graphs by examining the coordinate overlaps between the exons in the splicing graphs and the aligned long reads, similarly to the approach described in 'Merged alignments: creating splicing graph from both short and long reads' section.

For the purpose of this study, we implemented a naive approach that works as follows. For every long-read alignment and for every exon inferred from the short-read alignments, we calculate the overlap between the coordinates. Every exon $e$ that has an overlap with the read $r$ is considered a candidate for the path. Then, we look for a path $P = e_1, \ldots e_n$, where the starting coordinates of the candidate exons in the path are in ascending order. If no such path exists in the splicing graph, the path is considered to fail to align.

These concepts are illustrated in Figure 5E and F.

# Experiments

## Data and experimental setup

From human chromosome 2 (version GRCh38/hg38), we sampled all genes that fulfilled the following criteria: (i) shortest transcript was at least 1000 bases long, (ii) the longest transcript was at least 3200 bases long and (iii) there were at least two transcripts that did not share all of their inner borders (i.e. used the same exons with different 3′ or 5′ or both). There were 159 genes fulfilling these criteria. We randomly sampled 100 genes for testing.

For every chosen gene, we simulated 10 000 short (75 bp) reads with Illumina-like error profile, consisting of 1% substitution rate, and 1000 long reads of lengths {400, 800, 1200, 1600, 2000, 2400, 2800, 3200} with 11% insertion, 4% deletion and 1% substitution rate, which is reported as the error profile of PacBio reads [8].

For sampling the reads, we used RNASeqReadSimulator [12]. As the number of the long reads was low, we considered the transcripts to have uniform expression levels to guarantee that all the transcripts would get sufficiently sampled. That is, for every read, each transcript had an equal chance to be chosen as the source, regardless of the length of the transcript. The location of the read along the chosen transcript was also sampled from uniform distribution. For read lengths longer than 1000 bases, if the sampled transcript was shorter than the read length, the full-length transcript was added to the data set.

The use of uniform expression levels deviates from a real scenario, but it works also as a sufficient benchmark to enable transcript sequence prediction to consider all observed splice variants. With nonuniform distribution, the sampling rate needs to be increased significantly to guarantee the low-expressed transcripts to be present (Figure 10B). As our main focus in this article is exon chain prediction evaluation, the setting of uniform expression levels was chosen to optimize the number of observed splice variants being sampled to the test data.

For simulating Illumina-like error profile for the short reads, we used RNASeqReadSimulator. As RNASeqReadSimulator only simulates substitutions, for simulating PacBio-like error profile for the long reads, we used an in-house script. The insertions and deletions simulated by the script have length 1.

For comparison we also simulated both short and long reads without any errors; this gives a scale on how much the difficulty lies in the erroneous reads and how much in the spliced alignment problem.

For creating the splicing graphs, we used the software tool SpliceGrapher (version 0.2.4) [10]. As SpliceGrapher requires being given a gene model in advance, we gave it the annotated transcripts used for the simulation. In addition to the gene model, SpliceGrapher takes as an input a SAM format alignment file. For the alignment of the reads, we used GMAP (version 2014-10-22) [7].

For post-processing of the splicing graph, we used an in-house script to split any overlapping exons into pseudoexons, as explained in 'Merged alignments: creating splicing graph from both short and long reads' section. The script is trivial: all the start and end coordinates of the exons were collected, and based on these coordinates, the pseudoexon list was built.

As described in the previous section, we tested four different approaches: 1) creating a splicing graph from both aligned short and long reads, 2) creating a splicing graph from aligned short reads, and using dynamic programming to align the splicing graph and the long-read sequences converted into graphs, 3) creating a splicing graph from aligned short reads, and inferring the path by calculating overlaps between the nodes in the splicing graph and the genomic locations of the long-read alignments and 4) error-correcting the long reads using short reads, aligning only long reads and using these alignments to create a splicing graph.

For Cases 1, 2 and 3, we had two subcases, one for reads with simulated sequencing error, and one for reads without simulated error. Additionally, we experimented on using error-correction method as preprocessing step and then applying Case 2.

To examine how the correctness of the paths affects the downstream analysis, we assembled the transcripts from these data sets using software StringTie (version 1.0.1) [5] and Traphlor [4]. Both StringTie and Traphlor use minimum-cost flows to choose paths in the splicing graph.

StringTie has an integrated mechanism to predict transcripts directly from reads, while Traphlor can make use of the exon chains. The hypothesis is that the better the exon chains the better the Traphlor should perform in comparison with StringTie.

We attempted the downstream analysis also using Cufflinks [13], SLIDE [14], IsoLasso [15] and FlipFlop [6]. But for the bigger read lengths with realistic error profiles, we were either unable to run the tools because of memory allocation errors (Cufflinks, FlipFlop) or the tools did not produce any output (SLIDE, IsoLasso).

During our experiments, we found that SpliceGrapher sometimes creates many erroneous nodes (e.g. a gene model having only 8 nodes resulted in 306 nodes in the predicted graph when using both long- and short-read alignments). Traphlor's original problem formulation requires covering all the nodes in the flow network, but because of this problem, we relaxed the constraint to only require covering all the nodes corresponding to exon chains.

## Validation criteria

For validation, we created the 'ground truth' paths for all long reads by converting long-read BED files created by RNASeqReadSimulator to alignment BAM files using BEDTools [16], and mirroring these alignments on the splicing graph.

For a node to be added to the path, either the start and end coordinates for some block in the alignment had to match the start and end coordinates of the node, or the node had to be completely contained in some block in the alignment. This criteria was relaxed for the first and last block of the read to only require the end (respectively, start) to match the coordinates of the node. For the rest of the article, we refer to these paths as 'true paths'.

For Cases 1 (using both short and long reads) and 4 (error-correcting long reads), we mirrored the aligned long reads on the graph in the same fashion. Cases 2 (graph alignment) and 3 (calculating overlaps) directly produced paths. For the rest of the article, we refer to these paths as ‘predicted paths’.

For all cases, if a sequence of nodes would violate the structure of the graph, in such way that for any two nodes in the path, there was no arc between them in the splicing graph, the path was reported to fail.

Two paths were considered to match if they consisted of exactly the same sequence of nodes.

We define ‘sensitivity’ as the number of matched paths divided by the number of true paths, and ‘precision’ as the number of matched paths divided by the number of successfully predicted paths (i.e. excluding the predicted paths that reported to fail). If all the reads successfully produced paths, sensitivity = precision. Otherwise, precision can be higher than sensitivity. *F*-score, the standard measure of performance, is the harmonic mean of sensitivity and precision.

For the second part of our experiments, assembling transcripts, we used the same validation criteria as used in [4]. All predicted transcripts were matched against all the transcripts from which the reads were sampled. Two transcripts consisting of more than one exon were considered a match if all internal boundaries of the transcripts were identical (i.e. the beginning of first exon and the end of last exon did not need to match). Single-exon transcripts were considered to match if the overlapping area occupied at least 50% of the length of each transcript. Only one predicted transcript can match a single annotated transcript.

For this experiment, we define ‘sensitivity’ as the number of matched transcripts divided by the number of annotated transcripts and ‘precision’ as the number of matched transcripts divided by the number of predicted transcripts.

## Analysis of experiments

As can be seen in the Figure 6, for the data with realistic error profiles, both sensitivity and precision were highest using error-correction (Case 4) and overlap methods (Case 3). The best performance was found combining these two methods, which shows that the creation of the splicing graph from the short reads is still slightly more accurate than using error-corrected long reads. As the difference was not significant, we can with high probability assume that the difference was not because of low coverage of the long reads, that is the coverage was sufficient for the long reads to cover every splicing site.

Surprisingly, dynamic programming approach reached only half of the sensitivity and precision of the top performers, and, additionally, both of the measures decreased as read length increased. Although dynamic programming is guaranteed to find some optimal solution for each long read, for read alignment, this approach has one severe limitation; if the last base of the exon is the same as the last base of the intron, and both of them are incorporated into the splicing graph, choosing either will give score-wise optimal solution. Whereas for the purpose of validation, only the case that uses the last base of the exon is correct. For the reference, read aligners generally are able to use the information about canonical dinucleotides to break these kind of ties.

Also surprisingly, our baseline method, using both long and short reads in the creation of the splicing graph, had only slightly worse sensitivity and significantly higher precision than dynamic programming approach with read length of 400 bases. As a reminder, precision was defined as the number of matched paths divided by the number of successfully predicted paths. This result is likely because of a large portion of the reads failing to produce paths.

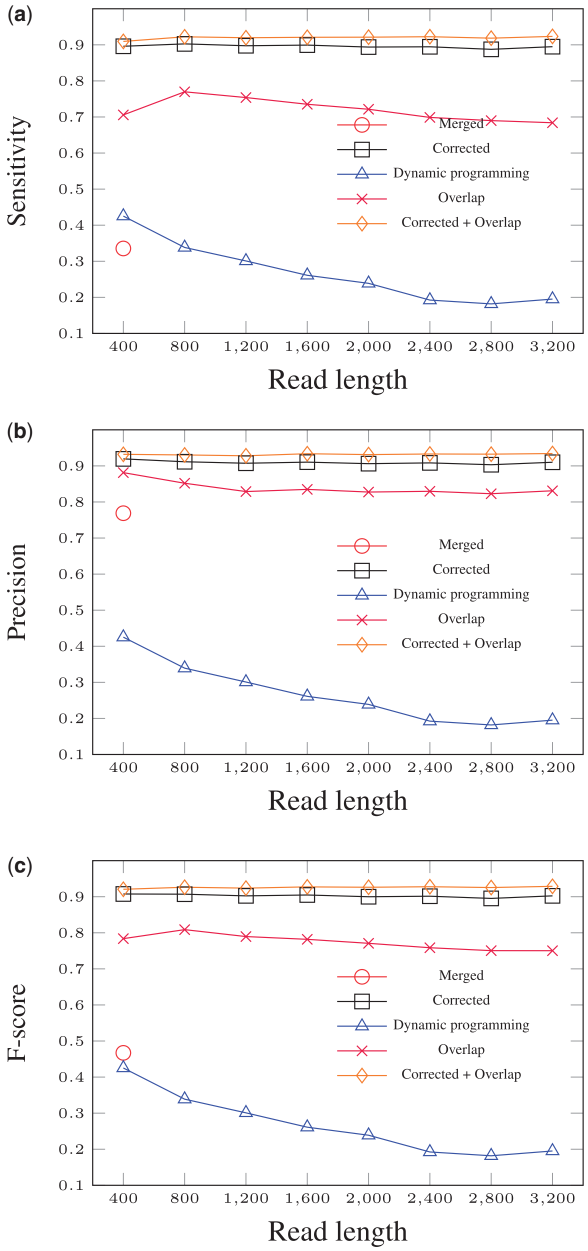


**Figure 6.** Exon chain prediction accuracy for the cases with 16% sequencing error. In ‘merged’ case, long-read alignments are mirrored on a graph made from both short and long reads, in ‘dynamic programming’, dynamic programming is used to align long reads on the splicing graph and in ‘overlaps’, the best overlap in genomic coordinates on the exons predicted from short reads is chosen. In ‘corrected’, the long reads with 16% sequencing error are first error-corrected with short reads, then aligned to the reference and these alignments are mirrored on graph made from short reads. ‘Corrected + overlap’ cases first use error-correction and then use the overlaps between short- and long-read alignments to infer the exon chains.

For the cases without errors (Figure 7), our baseline method achieved sensitivity >90%, on par with the method using overlaps to infer paths. Dynamic programming achieved ~10% higher sensitivity and precision than in the case with errors, but

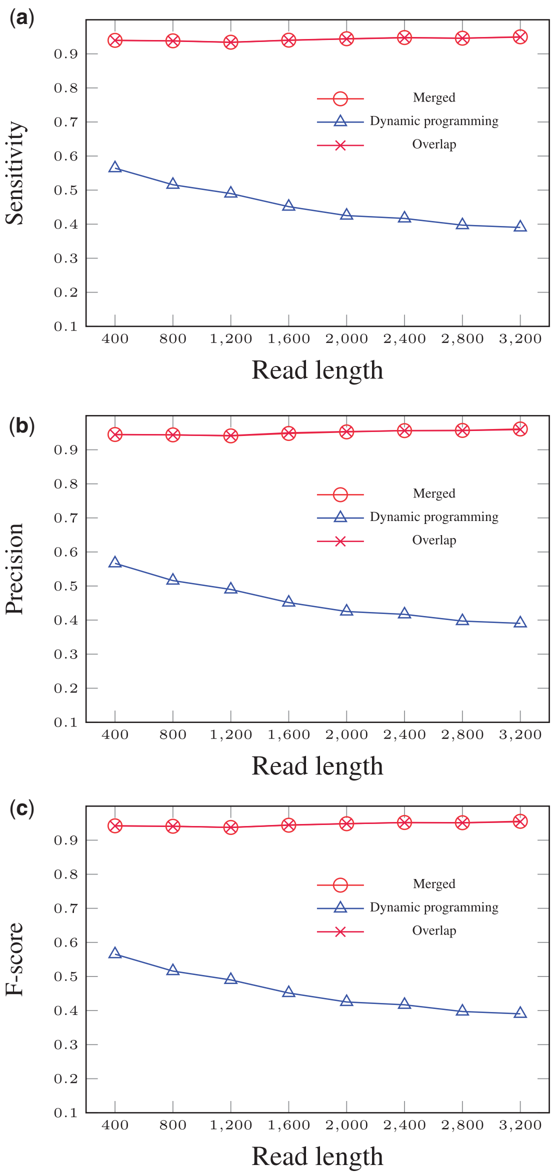


**Figure 7.** Exon chain prediction accuracy for the cases without sequencing errors. In 'merged' case, long-read alignments are mirrored on a graph made from both short and long reads, in 'dynamic programming', dynamic programming is used to align long reads on the splicing graph and in 'overlaps', the best overlap in genomic coordinates on the exons predicted from short reads is chosen.

was still significantly worse than the baseline method and the overlap method.

Owing to computational resource constraints, for the data set with simulated sequencing errors, we could only test using both long and short reads in the creation of the splicing graph (the baseline method) for read length of 400 bases. For longer read lengths, the creation of the splicing graph from the highly erroneous long reads took in the excess of 100 h per gene (i.e. processing the whole data set for one read length would have taken >13 months).

We also measured the running time (Table 1) and peak memory usage (Table 2) of the different approaches and parts of the pipeline. As expected, the time required for aligning the long reads and error-correcting them increases as read length increases, as does the time required for building the splicing graph from the corrected reads. Surprisingly, the time requirement for splicing graph creation did not grow significantly with the read length when using both short and long reads without sequencing errors. Excluding the splicing graph creation for both short and long reads, the main bottleneck for the pipeline was the dynamic programming module.

As memory testing takes several times the normal module execution time, we were unable to test dynamic programming memory requirements for larger read lengths. As the main memory requirement is building the matrix that takes $O(nm)$ space, where $n$ is the length of the reference and $m$ is the length of the read, the theoretical peak memory requirement doubles as read length doubles.

Although the memory tests were executed on a computing cluster node with 32 GB of RAM, Table 2 shows that all the modules tested fit into 4 GB of RAM, and as such are useable on a regular desktop machine.

For the downstream analysis, we used StringTie and Traphlor (Traphlor base) on the data set containing both short- and long-read alignments, and, additionally, gave the splicing graphs and paths created by the various modules to the flow engine of Traphlor. As seen in the Table 3, for the data with sequencing errors, the running times of both the original Traphlor and Traphlor's flow engine are linear in read length, but for some reason, StringTie's running time increases by an order of magnitude for every increase in read length. Executing StringTie on 1600 bp was aborted after 3 central processing unit (CPU) days had passed, with the assumption that the process had stalled. However, if the order of magnitude increase keeps up, the expected running time is almost 9 CPU days. However, for the data without sequencing errors, the running times are approximately constant for all approaches. We also tested StringTie with the preprocessing step of error correcting the long reads, and this input was much more manageable for it compared with the original input.

For the transcript predictions based on data with 16% sequencing error, the overall performance of almost all approaches at small read lengths was low (Figure 8), with sensitivity in the range of 20–40% and precision staying <40%. Exception to this was StringTie ran on error-corrected reads, which had precision at ~55%. When read length increased to 3200 bp, sensitivity for all the methods except Traphlor base and the exon chain approached based on dynamic programming increased to 60–80% range.

For the predictions based on the data without sequencing errors (Figure 9), performance of all the methods increased significantly as expected. The approach where exon chains were inferred from overlaps performed best when no sequencing errors were involved, beating StringTie in both sensitivity and precision.

Our hypothesis was that the correctness of the exon chains correlates with the accuracy of the transcript prediction, and based on Figures 8 and 9, the hypothesis seems to hold; dynamic programming approach had the worst correctness of the exon chains, and also has the worst transcript prediction accuracy on all three measures. Error-corrected reads alone, overlaps and error-correction + overlaps approaches have similar performances to each other, with varying rankings between the different measures.

Our baseline methods, Traphlor base and StringTie, ranked as expected. Traphlor base, which contains no heuristics for dealing with errors near splice sites, was systematically either the worst competitor or tied with the dynamic programming approach. StringTie, on the other hand, was among the best. StringTie was beaten by the overlap approach in sensitivity for both data with and without sequencing errors, and in precision

**Table 1.** The median running time per gene (in seconds) of different approaches and parts of the pipeline

| Case | 400 bp | 800 bp | 1200 bp | 1600 bp | 2000 bp | 2400 bp | 2800 bp | 3200 bp |
|---|---|---|---|---|---|---|---|---|
| With sequencing error | | | | | | | | |
| Short-read alignment | 4.46 | 6.15 | 7.73 | 9.66 | 8.33 | 8.71 | 8.34 | 7.71 |
| Long-read alignment | 6.34 | 18.09 | 31.39 | 43.02 | 44.99 | 59.57 | 64.07 | 61.68 |
| Error-correction | 10.95 | 37.33 | 76.45 | 124.28 | 133.79 | 189.38 | 216.78 | 195.82 |
| SpliceGrapher short reads | 11.28 | 12.64 | 13.07 | 13.62 | 12.81 | 13.35 | 12.50 | 13.83 |
| SpliceGrapher merged | 492.19 | – | – | – | – | – | – | – |
| SpliceGrapher corrected | 14.73 | 25.80 | 38.22 | 61.36 | 65.46 | 96.57 | 127.57 | 97.88 |
| Predicted paths in merged | 2.16 | – | – | – | – | – | – | – |
| Predicted paths in corrected | 1.39 | 3.01 | 5.16 | 8.05 | 9.71 | 12.73 | 14.42 | 14.25 |
| Dynamic programming | 5377.04 | 17 670.61 | 42 604.87 | 74 048.15 | 103 739.40 | 151 578.87 | 189 696.08 | 166 191.76 |
| Overlaps | 0.55 | 0.92 | 1.13 | 1.41 | 1.48 | 1.95 | 1.96 | 2.10 |
| Without sequencing error | | | | | | | | |
| Short-read alignment | 4.14 | 5.89 | 7.17 | 7.48 | 8.07 | 8.93 | 7.85 | 6.27 |
| Long-read alignment | 1.32 | 1.81 | 2.64 | 3.59 | 3.63 | 4.67 | 5.13 | 4.77 |
| SpliceGrapher short reads | 11.04 | 13.28 | 12.84 | 14.17 | 13.14 | 13.45 | 12.60 | 13.03 |
| SpliceGrapher merged | 12.56 | 11.28 | 11.66 | 12.40 | 12.88 | 12.57 | 12.32 | 13.20 |
| Predicted paths in merged | 0.76 | 1.32 | 1.91 | 2.29 | 2.74 | 2.55 | 2.15 | 2.20 |
| Dynamic programming | 4900.66 | 14 854.51 | 37 694.78 | 62 319.78 | 91 721.54 | 132 044.72 | 152 728.47 | 155 836.54 |
| Overlaps | 0.37 | 0.48 | 0.50 | 0.57 | 0.57 | 0.56 | 0.55 | 0.54 |

*Note.* 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. 'Merged' data set includes both short and long reads.

**Table 2.** The peak memory usage in megabytes for a randomly picked small subset of the data ($n$=5), as function of the read length

| Case | 400 bp | 800 bp | 1200 bp | 1600 bp | 2000 bp | 2400 bp | 2800 bp | 3200 bp |
|---|---|---|---|---|---|---|---|---|
| Short-read alignment | 653 | 723 | 715 | 664 | 624 | 668 | 631 | 684 |
| Long-read alignment | 798 | 989 | 1095 | 1145 | 1196 | 1150 | 1231 | 1199 |
| Error-correction | 341 | 341 | 341 | 341 | 342 | 341 | 341 | 341 |
| SpliceGrapher (merged alignments) | 3802 | – | – | – | – | – | – | – |
| SpliceGrapher (short reads) | 3802 | 2538 | 3802 | 2538 | 2538 | 2538 | 2538 | 2538 |
| SpliceGrapher (corrected reads) | 3802 | 2538 | 3802 | 3802 | 2538 | 3802 | 2538 | 3803 |
| Find paths (merged alignments) | 10 | – | – | – | – | – | – | – |
| Find paths (corrected reads) | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| Dynamic programming | 1184 | 2368[a] | 3552[a] | 4736[a] | 5920[a] | 7104[a] | 8288[a] | 9472[a] |
| Overlaps | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |

*Note.* For this test, we used the data sets with errors only.

[a] Dynamic programming values are estimates based on the theoretical bounds. The time for memory testing for them was unrealistic.

**Table 3.** The total CPU time of running the transcript prediction for all 100 genes

| Case | 400 bp | 800 bp | 1200 bp | 1600 bp | 2000 bp | 2400 bp | 2800 bp | 3200 bp |
|---|---|---|---|---|---|---|---|---|
| StringTie | 28 s | 24 min 22 s | 254 min 53 s | – | – | – | – | – |
| StringTie no error | 4 s | 5 s | 5 s | 5 s | 5 s | 6 s | 6 s | 6 s |
| StringTie corrected | 2 s | 3 s | 3 s | 3 s | 3 s | 3 s | 4 s | 4 s |
| Traphlor base | 2 min 3 s | 5 min 51 s | 13 min 26 s | 21 min 27 s | 26 min 12 s | 34 min 9 s | 42 min 15 s | 50 min 18 s |
| Traphlor base no error | 21 s | 25 s | 29 s | 32 s | 35 s | 38 s | 41 s | 44 s |
| Traphlor merged | 29 s | – | – | – | – | – | – | – |
| Traphlor merged no error | 5 s | 6 s | 7 s | 8 s | 9 s | 10 s | 10 s | 11 s |
| Traphlor corrected | 9 s | 17 s | 21 s | 25 s | 18 s | 20 s | 20 s | 18 s |
| Traphlor dp | 18 s | 58 s | 1 min 11 s | 2 min 23 s | 2 min 39 s | 3 min 31 s | 3 min 7 s | 4 min 27 s |
| Traphlor dp no error | 4 s | 4 s | 5 s | 5 s | 5 s | 5 s | 6 s | 5 s |
| Traphlor overlap | 5 s | 7 s | 8 s | 8 s | 6 s | 8 s | 8 s | 6 s |
| Traphlor overlap no error | 3 s | 3 s | 4 s | 4 s | 4 s | 4 s | 4 s | 4 s |

*Note.* Note that for StringTie and Traphlor base, the time includes creating the graphs. StringTie ran on the data with sequencing errors stalled with 1600 bp, and the merged graph for the data with sequencing errors was only created for 400 bp data set.

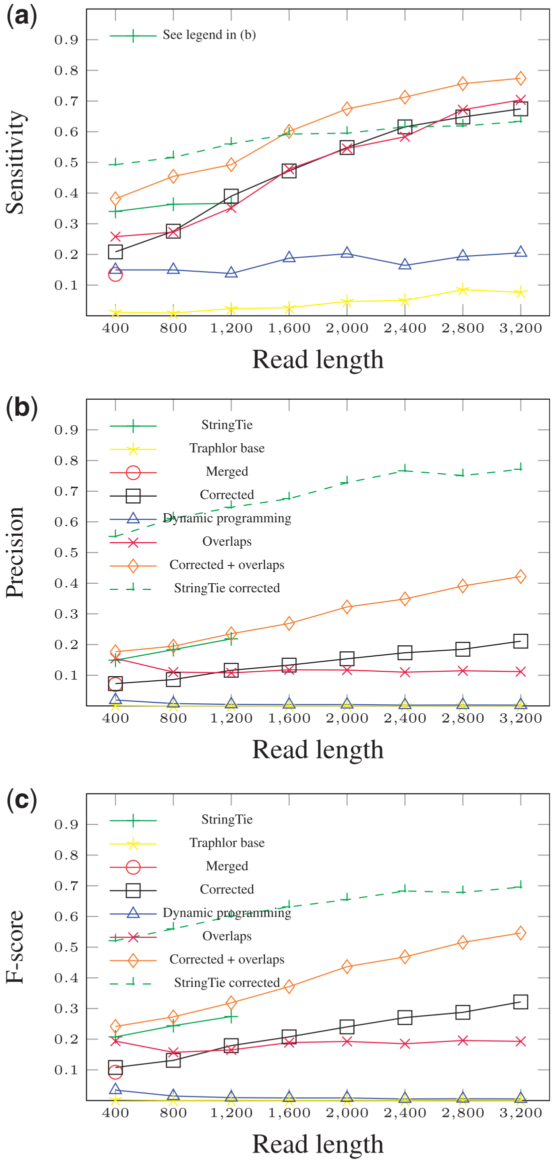


**Figure 8.** Transcript prediction accuracy using the data sets with 16% sequencing error. Transcripts were predicted from the alignment file (both short and long reads) using software StringTie and Traphlor (Traphlor base), which build their own graphs. In the remaining cases, the flow network module of Traphlor was given the predicted graphs and exon chains for each exon chain finding approach. We also tested StringTie with the preprocessing step of error correcting the long reads.

for the data without sequencing errors. However, after running the error-correction preprocessing step, performance of StringTie improved significantly.

## Discussion

In this article, we surveyed multiple approaches on finding exon chains, and experimented on the effectiveness of the approaches in finding the exon chains with different read lengths. We also experimented on the effect of the correctness of the exon chains on the downstream analysis (transcript prediction). Additionally, we examined the time and memory requirements of the various parts of the pipeline to identify possible bottlenecks.

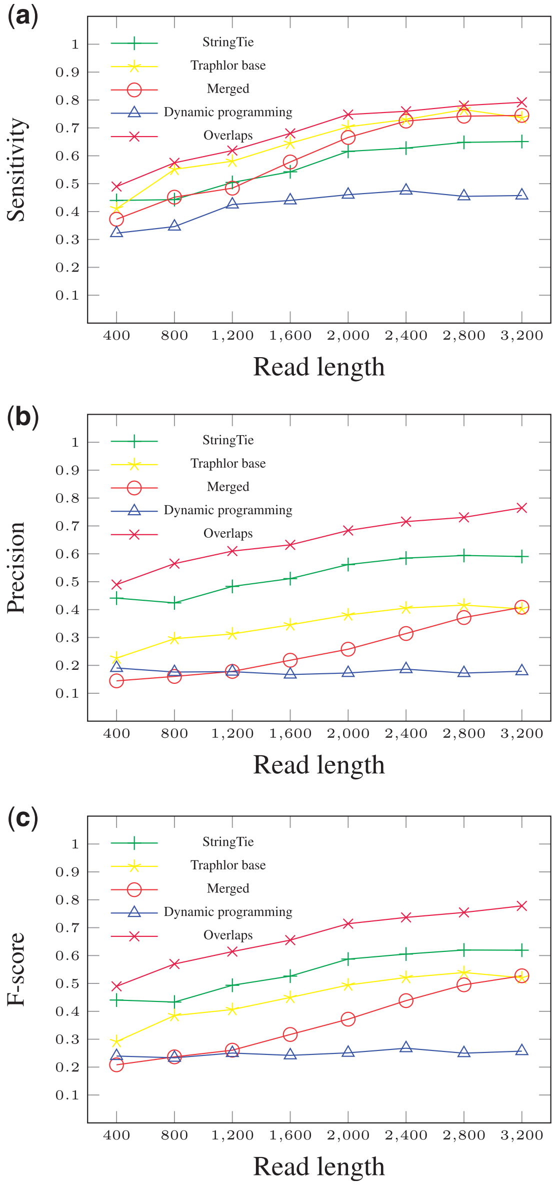


**Figure 9.** Transcript prediction accuracy using the data sets without sequencing errors. Transcripts were predicted from the alignment file (both short and long reads) using software StringTie and Traphlor (Traphlor base), which build their own graphs. In the remaining cases, the flow network module of Traphlor was given the predicted graphs and exon chains for each exon chain finding approach.

For exon chain prediction accuracy, using short reads to correct the long reads was clearly superior both in sensitivity and precision. It is noteworthy that the error-correction software that was used is not even tailored for RNA-seq reads, and so the error-correction results are likely to improve in the future.

Using short reads to create the splicing graph, and considering the overlaps between the coordinates of the splicing graph and coordinates of the long reads aligned to the reference genome also performed well. As expected, in the tests without simulated error, it was on par with the *F*-score of the error-correction method. However, simply using both short and long reads without sequencing error and creating the splicing graph from them performed at the same level.

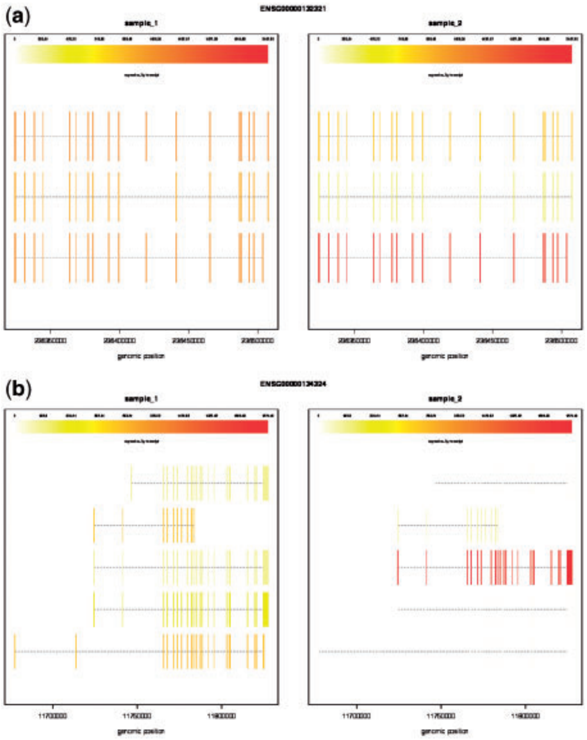


**Figure 10.** Two scenarios in differential expression analysis. (**A**) The pipeline finds the same transcripts when using uniform expression levels (left) and non-uniform expression levels (right). (**B**) The pipeline finds all the sampled transcripts when using uniform expression levels (left), but only two when using nonuniform expression levels (right).

Combining error-correction and examining the overlaps improved the results slightly over only using error-correction, which points to the graph creation from error-corrected reads not being as good as creating it from the short reads.

The hypothesis before conducting this study was that the better the exon chain prediction is the better the transcript prediction will be. Based on our experiments, this hypothesis seems to hold; dynamic programming was by far the worst approach for finding the exon chains, and the transcript prediction accuracy mirrors this. Also, the performance of the best approaches in finding exon chains mirrors to their performance in transcript prediction.

During the experiments, we observed that the splicing graph creation tool we used, SpliceGrapher, did not scale well with increasing read length using highly erroneous data when given both short and long reads. However, the tool did not have such scaling problems with using reads without sequencing error or the error-corrected long reads. It seems that the tool is not equipped to handle the high error rate.

Additionally, we found that SpliceGrapher sometimes creates many erroneous nodes. Although these low-confidence nodes are listed as ‘putative’ and could in theory be discarded, discarding all putative nodes could discard real novel exons and splicing events.

Based on our study, we think that for the future work, the most important direction is to develop a splicing graph creation tool that is both faster and better able to deal with the error profile of third-generation sequencing data.

Although dynamic programming did not show promise in this study, based on the results of using overlaps, we also believe an approach combining graph alignment with smart splicing site detection could be feasible, using e.g. colinear chaining (colinear chaining on graphs being a topic of another manuscript). Also, we used a fixed scoring scheme for the dynamic programming; one could easily improve the results with some parameter estimation.

One of the approaches we tested—error correcting the long reads with short reads—was shown to be applicable also beyond the exon chain optimization strategy; StringTie software performance on transcript prediction was boosted even more clearly than the approach using explicit exon chains.

Here, we used transcript sequence prediction to evaluate the different approaches to find exon chains. There are many approaches in quantifying transcript abundances [17] and differential expression analysis between samples [18–20] that can be applied to the output of transcript sequence prediction. To help the reader to apply such methods on top of the pipelines developed in this article, we have included in our scripts an example of running StringTie with error-corrected long reads with uniform and nonuniform expression levels on transcripts of a selected gene, and then comparing the expression-level differences. Figure 10 illustrates two different outcomes; with realistic expression-level differences, typically, some splice events are not sufficiently sampled, and not all transcripts are found. Among the 100 genes we tested, result as in Figure 10B was common.

### Key Points

- Transcript prediction can be modeled as a graph problem, where reads spanning several exons can be modeled as exon chains.
- Third-generation sequencing produces significantly longer reads than second-generation sequencing, but the error rate is an order of magnitude higher.
- Modeling long reads as exon chains spanning three or more exons can improve transcript prediction accuracy significantly, but the high error rate makes them difficult to align reliably around splice sites.
- We identify practical exon chain identification scenarios to significantly improve the preprocessing steps of transcript prediction tools Traphlor and StringTie.
- The simulated data and in-house scripts provided with this article give analysis workflow developers an opportunity to better exploit third-generation long-read RNA sequencing data.

## Acknowledgements

The authors thank the anonymous reviewers for constructive comments that helped them to improve the presentation. The PacBio ‘Reads of Insert’ technique as a plausible way to lower the error rate was hinted to them by an anonymous reviewer.

## Funding

The Academy of Finland (grant number 284598 for Center of Excellence in Cancer Genetics Research).

## References

1. Sharon D, Tilgner H, Grubert F, *et al*. A single-molecule long-read survey of the human transcriptome. *Nat Biotechnol* 2013;**31**(11):1009–14.
2. Heber S, Alekseyev M, Sze SH, *et al*. Splicing graphs and EST assembly problem. *Bioinformatics* 2002;**18(Suppl 1)**:S181–8.
3. Rizzi R, Tomescu AI, Mäkinen V. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinformatics* 2014;**15**(S-9):S5.
4. Kuosmanen A, Sobih A, Rizzi R, *et al*. On Using Longer RNA-seq Reads to Improve Transcript Prediction Accuracy. In *8th International Conference on Bioinformatics Models, Methods, and Algorithms*. ScitePress, 2016.
5. Pertea M, Pertea GM, Antonescu CM, *et al*. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nat Biotechnol* 2015;**33**(3):290–5.
6. Bernard E, Jacob L, Mairal J, *et al*. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics* 2014;**30**(17):2447–55.
7. Wu TD, Watanabe CK. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics* 2005;**21**(9):1859–75.
8. Chaisson MJ, Glenn T. Mapping single molecule sequencing reads using Basic Local Alignment with Successive Refinement (BLASR): theory and application. *BMC Bioinformatics* 2012;**13**:238.
9. Salmela L, Rivals E. LoRDEC: accurate and efficient long read error correction. *Bioinformatics* 2014;**30**(24):3506–14.
10. Rogers MF, Thomas J, Reddy AS, *et al*. SpliceGrapher: detecting patterns of alternative splicing from RNA-Seq data in the context of gene models and EST data. *Genome Biol* 2012;**13**(1):R4.
11. Mäkinen V, Belazzougui D, Cunial F, *et al*. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015.
12. Li W. http://alumni.cs.ucr.edu/∼liw/rnaseqreadsimulator.html 2012.
13. Trapnell C, Williams BA, Pertea G, *et al*. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat Biotechnol* 2010;**28**(5):511–5.
14. Li JJ, Jiang CR, Brown JB, *et al*. Sparse linear modeling of next-generationm RNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proc Natl Acad Sci USA* 2011;**108**(50):19867–72.
15. Li W, Feng J, Jiang T. IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. *J Comput Biol* 2011;**18**(11):1693–707.
16. Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* 2010;**26**(6):841–2.
17. Li B, Dewey CN. REEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics* 2011;**12**:323.
18. Robinson MD, McCarthy DJ, Smyth GK. A bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 2010;**26**(1):139–40.
19. Anders S, Huber W. Differential expression analysis for sequence count data. *Genome Biol* 2010;**11**(10):R106.
20. Glaus P, Honkela A, Rattray M. Identifying differentially expressed transcripts from RNA-seq data with biological variation. *Bioinformatics* 2012;**28**(13):1721–8.