IsoTree: A New Framework for De novo Transcriptome Assembly from RNA-seq Reads

Jin Zhao, Haodi Feng*, Daming Zhu, Chi Zhang, and Ying Xu

Abstract—High-throughput sequencing of mRNA has made the deep and efficient probing of transcriptome more affordable. However, the vast amounts of short RNA-seq reads make de novo transcriptome assembly an algorithmic challenge. In this work, we present IsoTree, a novel framework for transcripts reconstruction in the absence of reference genomes. Unlike most of de novo assembly methods that build de Bruijn graph or splicing graph by connecting k-mers which are sets of overlapping substrings generated from reads, IsoTree constructs splicing graph by connecting reads directly. For each splicing graph, IsoTree applies an iterative scheme of mixed integer linear program to build a prefix tree, called isoform tree. Each path from the root node of the isoform tree to a leaf node represents a plausible transcript candidate which will be pruned based on the information of paired-end reads. Experiments showed that in most cases IsoTree performs better than other leading transcriptome assembly programs.

IsoTree is available at https://github.com/Jane11011107/IsoTree.

Index Terms—RNA-seq, de novo assembly, alternative splicing, transcriptome.

1 INTRODUCTION

Alternative splicing occurs as a normal phenomenon in eukaryotes, where it greatly increases the diversity of proteins that can be encoded by the genome [1]. A recent study estimated that more than 95% of all multi-exon genes are alternatively spliced [2]. Besides that, numerous researches have revealed that a great deal of human diseases, especially cancer, are related to abnormal splicing [3], [4]. Advances in RNA-seq have opened the door to efficient probing of full-length transcripts. RNA-seq technology can generate hundreds of millions of short reads (50 – 250bp) from sequenced transcripts (complete and contiguous mRNA sequence from the transcription start site to the transcription end site, for multiple alternatively spliced isoforms). Despite the opportunities for transcriptome assembly, great challenge emerges of how to subtly recover as many expressed transcripts as possible at lowest cost from massive short reads. Many obstacles remain in transcriptome reconstruction such as sequencing bias or sequencing error, variable sequence coverage, alternative transcripts from the same locus sharing the same exons, and the existence of paralogs genes. A successful method should address these issues, and apply a suitable data structure to accommodate multiple transcripts per locus.

A growing number of strategies have been developed to solve the transcriptome assembly problem based on RNA-seq. They can be generally divided into two categories: genome-based and de novo assembly approaches. Genome-based approaches, such as Cufflinks [5], Scripture [6], Bayesember [7], IsoInfer [8], IsoLasso [9], Traph [10], iReckon [11], CIDANE [12], StringTie [13], TransComb [14], and Scallop [15] usually first align the reads to a reference genome with alignment tools such as TopHat [16], TopHat2 [17], GSNAP [18], STAR [19], and SpliceMap [20], and then merge sequences from the same gene locus according to overlapping alignments and splicing junctions to build a graph representing all possible isoform transcripts. Finally, different models are adopted to recover full-length transcripts from the graph. For example, Cufflinks applies the minimum-cost path cover model, StringTie employs a network flow algorithm originally developed in optimization theory, and Traph uses minimum-cost flow model combined with a greedy algorithm. However, the reference genome especially a cancer genome is not always available. In these situations, de novo assembly is required. In theory, a de novo assembler can reconstruct transcripts even on regions that are missing a reference.

The field of de novo assembly developed from pioneering work on de Bvuijn graphs [21], [22], in which a vertex is a k-mer and an edge exists between two vertices u and v if and only if u and v appear consecutively in a read. Simple paths in such graphs usually represent fragments of transcripts. However, de Bvuijn graphs may be very tangleome and therefore hard to deal with. Splicing graphs emerged at the right moment, which are more tractable than de Bvuijn graphs. A splicing graph of a locus is a directed acyclic graph, whose vertices represent exons while edges correspond to splicing junctions. To summarize, Trinity [23], ABySS [22], and IDBA-Tran [24] take the advantage of de Bvuijn graphs approach, while Bridger [25] and BinPacker [26] apply the splicing graphs strategy.

Trinity [23] plays a milestone role in de novo transcriptome assembly. It assembles transcripts by first extending contigs greedily, then building de Bvuijn graphs from these
contigs, and then extracting sufficiently covered paths from these graphs to construct splicing variants based on a brute-force enumeration strategy. BinPacker is a recently developed method, which searches for an optimal edge-path-cover over the splicing graphs by iteratively solving a series of bin packing problems. In this scheme, it uses a heuristic algorithm to update trajectories of items (edge-path-cover), and the iteration will terminate within $O(|V|)$ times (where $V$ is the node set of the splicing graph).

So far, all the existing de novo assemblers usually extend the contigs by connecting $k$-mers (a set of $k$-length substrings arising from read sequences). The $k$-mer-based contig extending strategy benefits from the speed of finding $(k-1)$ overlapping candidate $k$-mers, with the cost of losing information of the whole nucleotides arrangement in a read. Given that larger $k$ values tend to perform better on higher expressed transcripts or on longer transcripts while smaller values are more suitable for reconstructing lower expressed transcripts and shorter transcripts, some assemblers apply a multiple-$k$ strategy, such as ABySS [22], Oases-M [27], and IDBA-Tran [24].

In this paper, we present IsoTree, a de novo transcriptome assembler. The central idea behind IsoTree is to subtly extract transcript-representing paths from the splicing graph. The splicing graph is constructed with contigs that are extended by reads directly. Each vertex as well as each edge in the splicing graph is weighted by reads per base. IsoTree converts the splicing graph to a prefix tree by calling $(|V| - 2)$ times of a variant mixed integer linear program model with the objective to seek as few transcripts in the model as possible under the coverage constraints (see Methods for details).

We tested the performance of IsoTree on both simulated datasets and real datasets. On simulated datasets, we not only compared IsoTree to state-of-the-art de novo assemblers including Trinity [23], BinPacker [26], SOAPdenovo-Trans [28], IDBA-Tran [24], Oases [27], and Velvet [21], we also compared IsoTree with two widely used genome-based assemblers Cufflinks [5] and StringTie [13]. Scallop [15] is a newly developed genome-based strategy. Although the source code of Scallop is online, its guideline is not available and the software is hard to execute. Hence, we did not compare IsoTree with Scallop. For the real datasets, we compared IsoTree to de novo assemblers Trinity, BinPacker, SOAPdenovo-Trans, IDBA-Tran, Oases, and Velvet. Since accurate reference genomes were not available, we would not be able to compare the performance of IsoTree to genome-based assemblers. We employed blast+ [29] to evaluate the performance of each assembler. Our experiments demonstrated the superior performance of IsoTree on both simulated datasets and real datasets.

2 Methods

Splicing graph is originally put forward by Heber et al in 2002 [30]. Theoretically, each splicing graph constructed by IsoTree corresponds to an expressed gene: the vertices represent exons, the edges represent splicing junctions, and some paths correspond to isoforms generated by the gene. IsoTree applies a heuristic algorithm to convert the splicing graph into a prefix tree where each path from the root node to a leaf node represents a transcript. The expression level of a transcript relates to the weight of the leaf node in the corresponding isoform path. The general work flow of IsoTree algorithm is given in Figure 1.

2.1 Constructing splicing graph

IsoTree modified BinPacker’s splicing graph construction method [26]. Specifically, IsoTree extends contigs by reads while BinPacker extends them by $k$-mers.

In BinPacker’s contig extending strategy, the contig is extended by repeatedly selecting the most frequent $k$-mer that overlaps with the current contig terminus by its $k-1$ character prefix. However, the most frequent $k$-mer is not necessarily the most suitable $k$-mer. For example, suppose the contig sequence is “CCACGTGT”, and there are two $k$-mers (“GTTC” and “GTTA”) whose $k-1$-length prefix are exactly the $k-1$-length suffix of the contig. The frequency of “GTTC” is 3 that is contributed by reads “TTGTTCG”, “CGAGTTC”, and “CGTTCAG”. The frequency of “GTTA” is 2 that is contributed by reads “CTGTTGC”, “CGAGTTC”, and “CGTTGTC”. The frequency of “GTTA” is 2 that is contributed by reads “ACTGTAA” and “TGTTACG”. Obviously, “GTTA” is the better choice to extend the contig, because the connection is well supported by reads “ACTGTAA” and “TGTTACG”. However, BinPacker will choose “GTTC” to extend the contig with no read supporting the connection. By contrast, IsoTree extends contigs directly by reads that hold the longest overlaps to current contig terminuses.

IsoTree applies a hash table of $k$-mers to quickly determine reads that hold $x$-length overlaps to current contig. It decomposes each $Lbp$-length read sequence into $L - k + 1$ overlapping $k$-mers. For each $k$-mer, the hash table takes the $k$-mer sequence as key and the set of reads that contain this $k$-mer as value. The $k$-mer sequence is stored as a 64-bit unsigned integer with 2-bit nucleotide encoding. In the process of building a hash table, if the $k$-mer composed by the first $k$ nucleotides of a read appears at the first
time, the read is seen as a seed read. In order to get rid of sequencing errors, IsoTree excludes k-mers with low frequency following the criteria used by Trinity [23]. The read that contains the excluded k-mer is deleted.

IsoTree builds the splicing graphs following the next five steps.

**step 1:** Select an unused seed read as the main contig of the initial splicing graph.

**step 2:** Extend the main contig in two directions by repeatedly selecting an unused read with the highest priority from the candidate read set. If a candidate read has x-length \( (a \leq x \leq b, \text{ default } a = k, b = L - 1) \) overlaps with the current contig terminus, then its priority is \( x \).

The candidate reads that hold x-length overlaps to current contig terminus can be found quickly according to two k-mers in the current contig terminus. For forward extension (from 5' to 3'), set k-mer1 as the k-length suffix of the current contig. Suppose the value of k-mer1 in hash table is stored in an N-dimensional incremental vector \( R_1 \) with component \( R_1(i = 1, 2, \cdots, N) \) representing the ith read ID. In contrast, set k-mer2 as the k-mer that contains exactly the last \( x \)th to the last \( (x - k + 1)\)th letters of the current contig. Denote the M-dimensional incremental vector \( R_2 \) as the value of k-mer2 in hash table. If a read’s ID both belongs to \( R_1 \) and \( R_2 \), then IsoTree further checks if it has x-length overlaps with the current contig terminus. The algorithm to get the candidate reads for forward extension from sets \( R_1 \) and \( R_2 \) is described as Algorithm 1. Similarly, the candidate reads for reverse extension (from 3' to 5') can be found by another two k-mers. When a contig cannot be extended in either direction, it is used as the trunk node of a splicing graph to be constructed.

**Algorithm 1** Algorithm to get candidate reads for forward extension

```plaintext
1: int r1 = 1;
2: int r2 = 1;
3: while r1 <= N and r2 <= M do
4:   if \( R_1[i] = R_2[i] \) then
5:     if the x-length suffix of contig == the x-length prefix of read \( R_1[i] \) then
6:       put \( R_1[i] \) to candidate read set
7:   end if
8:   r1 = r1 + 1
9:   r2 = r2 + 1
10: else
11:   if \( R_1[i] > R_2[i] \) then
12:     r2 = r2 + 1
13:   else
14:     r1 = r1 + 1
15:   end if
16: end if
17: end while
```

**step 3:** For each read in the current splicing graph, check if it could be extended by unused reads. Such a read is called a junction read. Once IsoTree finds a junction read, IsoTree keeps extending it until encountering an already used read that exactly lies in the downstream of the junction read or it can make no further extension by using steps 1 to 2 (Figure 2). If the former occurs, then a new junction read is identified, and the current splicing graph is updated by merging their matched x nucleotides. For the latter case, IsoTree applies the following criteria to determine whether to add the new branch to the current splicing graph: (a) the branch is long enough \((\geq 80 \text{ bp})\) to be an exon; (b) the branch is not similar with the corresponding part of the trunk; (c) there are at least two reads supporting this branch. Repeat step 3 until all junction reads have been processed.

**step 4:** The graph is then trimmed with the similar idea in Trinity: (a) for each edge, there must be a minimal number of reads (default 2) matched on each side of the junction; (b) the coverage of each edge must exceed 0.04 times the average coverage of two flanking nodes (twice the sequencing error rate in a read, the upper bound is about 2%); (c) if there is a node with several outgoing edges, the coverage of each of them should be more than 5% of the total outgoing edge coverage; (d) the coverage of any outgoing edge should be more than 2% of the total incoming edge coverage. Edges that violate any of these criteria are removed. Each isolated vertex in the current graph will be removed if its length is smaller than the pre-defined minimum transcript length. Until now, a splicing graph is constructed.

**step 5:** If there are remaining unused seed reads, repeat steps 1 to 4.

2.2 Balancing splicing graph

Let \( G(V, E) \) represent the splicing graph. Each vertex in the graph corresponds to an exon sequence. The sequence of
edge \((u, v)\) consists of \(L - 1\)-length suffix of node \(u\) and \(L - 1\)-length prefix of node \(v\). In order to facilitate building isofore tree steps, IsoTree adds a source vertex \(s\) and a sink vertex \(t\) into the splicing graph, connects \(s\) to vertices without incoming edges, and connects vertices without outgoing edges to \(t\). IsoTree weights each node or edge sequence as the number of reads per base [13]. Specially, IsoTree assigns rules.

IsoTree balances this type of exon node with the following rules.

For each vertex \(v \in V^\prime\{s, t\}\), check whether its incident edges satisfy the following conditions:

\[
(1 - \varepsilon)W_{in}(v) \leq (1 + \varepsilon)W_{out}(v),
\]

\[
(1 + \varepsilon)W_{in}(v) \geq (1 - \varepsilon)W_{out}(v),
\]

where \(\varepsilon\) is an empirical value (default 0.3), \(W_{in}(v)\) and \(W_{out}(v)\) represent the weight sum of all the edges entering vertex \(v\) and the weight sum of all the edges leaving \(v\), respectively.

If edges incident with \(v\) do not meet the above conditions at the same time, it means that there is a huge gap between the weights of the incoming and outgoing edges of \(v\), and the balancing process is needed. The approach to balance vertex \(v\) is as follows:

if condition (1) is violated, then

\[
E = E \cup \{(v, t), W(v, t) = W_{in}(v) - W_{out}(v)\};
\]

if condition (2) is violated, then

\[
E = E \cup \{(s, v), W(s, v) = W_{out}(v) - W_{in}(v)\};
\]

where \(W(v, t)\) is the weight of edge \((v, t)\), and \(W(s, v)\) is the weight of edge \((s, v)\).

### 2.3 Building isofore tree and recovering transcripts

IsoTree iteratively calls a variant of mixed integer linear program to comb all the transcripts encoded in a splicing graph to a prefix tree, called isofore tree. A splicing graph is transformed into an isofore tree by the following steps.

**step 1**: IsoTree first sets source vertex \(s\) in the splicing graph \(G(V, E)\) as the root node of isofore tree \(T\), and set \(v = s\). Each vertex \(u \in V\) is set as the child node of \(v\) in the Tree \(T\), with weight \(W_T(u) = W(v, u)\).

For example, a splicing graph with eight nodes is shown in Figure 3.a. IsoTree first sets the source vertex \(s\) of the splicing graph as the root node of isofore tree \(T\), and sets vertex \(A\) as the child node of \(s\). (Figure 3.b).

**step 2**: For splicing graph \(G\), set \(v\) as \(v_R\) (where \(v_R\) is the right node of \(v\) in topological order). Suppose that there are total of \(N\) edges leaving vertex \(v\). Let \(\beta\) denote the weight of edge \((v, \gamma) \in E\) \((1 \leq j \leq N)\).

In the above example, the \(v\) is updated by \(A\). As shown in Figure 3.a, there are total of two edges leaving vertex \(A\), i.e., \((A, B)\) and \((A, C)\). Let \(y_1 = B, y_2 = C\). Then \(\beta_1 = 4.0, \beta_2 = 3.01,\) and \(N = 2\).

**step 3**: For isofore tree \(T\), search leaf nodes with label \(v\) and mark them as \(x_1, x_2, \cdots, x_M\) (where \(M\) is the total number of leaf nodes with label \(v\)). Set \(\alpha_i \leq i \leq M\) denote the weight of node \(x_i\) in the isofore tree. Obviously, \(\alpha_i\) must be the weight of an incoming edge of \(v\) in splicing graph \(G\) or the splitting weight of an incoming edge.

As shown in Figure 3.b, there is only one leaf node labeled \(v\), that is \((A, 0.97)\). Hence, \(x_1 = A, \alpha_1 = 0.97,\) and \(M = 1\).

**step 4**: Expand each leaf node \(x_i (1 \leq i \leq M)\) by \(y_j (1 \leq j \leq N)\). IsoTree formalizes it into an optimization problem: assign \(\alpha_1, \alpha_2, \cdots, \alpha_M\) to \(\beta_1, \beta_2, \cdots, \beta_N\) with the parsimony objective. Let \(M \times N\) matrix \(C\) be the assignments matrix, with component \(c_{ij}\) representing the value of \(\alpha_i\) assigned to \(\beta_j\) \((1 \leq i \leq M, 1 \leq j \leq N)\). If \(c_{ij} > 0\), vertex \(y_j\) will be added to the isofore tree \(T\) as a child of \(x_i\). The weight of node \(y_j\) in the isofore tree \(T\) is \(\beta_j c_{ij} / \sum_{i=1}^{M} c_{ij}\). The value of \(c_{ij}\) must satisfy the following constraints:

\[
0 \leq c_{ij} \leq (1 + \varepsilon) \alpha_i, \quad 1 \leq i \leq M, 1 \leq j \leq N,
\]

\[
0 \leq c_{ij} \leq (1 + \varepsilon) \beta_j, \quad 1 \leq i \leq M, 1 \leq j \leq N,
\]

Here, IsoTree introduces a binary integer variable \(z_{ij}\) specifying whether a child is added by the following constraints:
\[ z_{ij} \leq \lambda c_{ij} \quad 1 \leq i \leq M, 1 \leq j \leq N, \quad (5) \]
\[ c_{ij} \leq \lambda z_{ij} \quad 1 \leq i \leq M, 1 \leq j \leq N, \quad (6) \]

where \( \lambda \) is a large positive number. If \( c_{ij} = 0 \), from (5), we have \( z_{ij} = 0 \) indicating that node \( y_j \) is not a child of node \( x_i \) in \( T \). On the other hand, if \( c_{ij} > 0 \), from (6), we have \( z_{ij} = 1 \) indicating that node \( y_j \) is a child of node \( x_i \).

In order to avoid bias assignments and make sure that each node in \( x_1, \cdots, x_M \) has at least one child node and each vertex in \( y_1, \cdots, y_N \) has been added to the isoform tree, we have:

\[ (1 - \varepsilon)\alpha_i \leq \sum_{j=1}^{N} c_{ij} \leq (1 + \varepsilon)\alpha_i \quad 1 \leq i \leq M, \quad (7) \]
\[ (1 - \varepsilon)\beta_j \leq \sum_{i=1}^{M} c_{ij} \leq (1 + \varepsilon)\beta_j \quad 1 \leq j \leq N. \quad (8) \]

We minimize the sum of \( z_{ij} (1 \leq i \leq M, 1 \leq j \leq N) \) following the parsimony principle to seek as few transcripts in the prediction as possible:

\[ \min f = \sum_{i=1}^{M} \sum_{j=1}^{N} z_{ij}. \quad (9) \]

Equations (3)-(9) form a mixed integer linear program (MILP). We adopt the branch-and-bound algorithm to solve the mixed integer linear programming problems (a few softwares are available for solving MILP by using this strategy). Since the transcript sequence only contains ‘A’, ‘C’, ‘G’, and ‘T’ four kinds of nucleotides, in most cases the number of outgoing (or incoming) edges of an exon node is less than 3. Besides, the sum of nodes in a splicing graph corresponding to one locus is less than 10 in most cases. Hence, the variables in our MILP model are usually less than 20 and the MILP can be solved by the branch-and-bound algorithm fast.

Following the above example, IsoTree puts \( \alpha_3 = 6.97 \), \( \beta_1 = 4.0 \) and \( \beta_2 = 3.01 \) into the mixed integer linear program model. Through solving this mixed integer linear program model, IsoTree obtains the value of \( c_{ij} (i = 1, j = 1, 2) \), i.e., \( c_{11} = 4.0, c_{12} = 3.01 \). Then, IsoTree sets \( y_1 \) (that is \( B \)) and \( y_2 \) (that is \( C \)) as child nodes of \( x_1 \) (that is \( A \)) in the isoform tree (Figure 3c), and weighs nodes \( y_1 \) and \( y_2 \) with \( \beta_1 c_{11} / \sum_{i=1}^{1} c_{i1} = 4.0 \) and \( \beta_2 c_{12} / \sum_{i=1}^{1} c_{i2} = 3.01 \), respectively.

**step 5:** Repeat steps 2 to 4 until \( v = t \).

The isoform tree is built after calling \( |V| - 2 \) times MILP in splicing graph \( G(V, E) \). Each path from the root node to a leaf node in the isoform tree represents a potential transcript, and the weight of a leaf node can be seen as an approximation of the transcript expression level. If paired-end information is available, IsoTree will map the reads to potential transcripts, and a transcript will be discarded if the transcript is not well covered by the paired-end reads.

### 3 Results

We compared IsoTree with the state-of-the-art assembly programs including six de novo assemblers BinPack(0,version 1.0), Trinity (version 2.3.2), SOAPdenovo-Trans (version 1.03), IDBA-Trans (version 1.1.1), Oases (version 0.2.8), Velvet (version 1.2.10), and two genome-based assemblers Cufflinks (version 2.1.1) and StringTie (version 1.3.1) on both simulated and real datasets.

On simulated datasets, we run IsoTree and the other eight compared assemblers on samples with different read lengths to investigate the impact of read length on assemblers’ performances. We run the de novo assemblers with different \( k \)-mer lengths to detect the most suitable \( k \) value. We also evaluated assemblers for different transcript expression levels. All the simulated datasets and assembly results are available at [https://pan.baidu.com/s/1dGl990](https://pan.baidu.com/s/1dGl990).

On real datasets, we compared IsoTree with the other six de novo assemblers. We did not compare IsoTree to the two genome-based assemblers Cufflinks and StringTie, since we could not get accurate reference genomes of the real datasets. On real datasets, we first run the de novo assemblers with different lengths of \( k \)-mers, and then compare the performances of the de novo assemblers with their favorite \( k \)-mer lengths. We also evaluated the computational demands of the de novo assemblers for runtime time and memory.

All assemblers were run on a server with 256GB of RAM and E5-2620V3*2 CPU processor.

### 3.1 Datasets

Mimicking the characteristic of real RNA-seq data, we generated total of 21 datasets of simulated paired-end reads from 100 isoform transcripts originated from 41 different genes in chromosome 1 (CRCh38.83, NCBI) using FluxSimulator [31]. Each simulated dataset contains 0.1 million paired-end reads. All the simulated samples were generated from the same library. The only difference between these samples is the read length. The lengths of reads in these 21 samples fall in the scope of 500b \( \sim \) 150bp.

We collected dog dataset and human dataset from NCBI SRA database (https://www.ncbi.nlm.nih.gov/) as the real datasets. There are total of 30968059 paired-end reads in dog dataset (Accession Code: SRX295047). The length of reads in dog dataset is 50bp. The human dataset (Accession Code: SRR3692633) contains 43675886 paired-end reads with length of 75bp. Both the dog dataset and the human dataset were sequenced under Illumina HiSeq system. The annotation transcripts (referred to as "reference transcripts") for these two samples were downloaded from UCSC (https://genome.ucsc.edu/). Note that, the annotation transcripts are not the ground truth expressed transcripts. Usually for a given sample, only a small number of annotation transcripts are expressed, and some novel predicted transcripts may not be included in the annotation [15].

### 3.2 Evaluation criteria

In this paper, all assembled transcripts were aligned to reference transcripts by blast+ [29]. We applied identity (the extent to which two sequences have the same residues at the same positions in an alignment, often expressed as
3.3 Simulated data

We constructed a thorough and detailed analysis of the performances of IsoTree and the other assemblers on simulated datasets. The benefit of using the simulated data is that we can set up the length of sequenced reads and expression level of sequenced transcripts according to our needs. Besides, it is easy to obtain the exactly reference transcripts. In this section, we compared the performances of IsoTree and the other assemblers on 21 simulated datasets with different read lengths. We found the most suitable k-mer length for each assembler by experimenting with different values of k. Finally, we evaluated the best performance of assemblers on transcripts with different expression levels.

With the development of RNA sequencing technology, the sequenced reads become longer and longer. We run all the assemblers on 21 simulated samples whose read lengths fall in the scope of 50bp ~ 150bp (Figure 4).

From Figure 4, we conclude that IsoTree is more competitive than the other assemblers both with recall and precision measurements. We observed an interesting phenomenon that IsoTree, Trinity, and BinPacker always performed better than Cufflinks and StringTie on all simulated datasets. It is worth to mention that IsoTree, Trinity, and BinPacker are all de novo assemblers while Cufflinks and StringTie are genome-based approaches. This is beyond our expectation that genome-based approaches should perform better than de novo approaches. We speculate that the reason is that the reference genome (chromosome 1) is usually far longer than even the superstring of the sequenced transcripts, which increases the uncertainty of alignments. The de novo assemblers’ excellent performances confirmed the importance of developing de novo assembly method since genome-based approaches do not apply for all assembly situations even though the reference genome is given.

For the recall measure, IsoTree outperformed all the other assemblers on all simulated samples (Figure 4a). Averaged over the 21 simulated samples, IsoTree recovered 26.6%, 42.1%, 99.7%, 115.6%, 86.3%, 216.1%, 183.3%, and 209.8% more full-length identified transcripts than Trinity, BinPacker, SOAPdenovo-Trans, IDBA-Trans, Oases, Velvet, Cufflinks, and StringTie, respectively. Of all the other assemblers except IsoTree, Trinity obtained the highest recall on the simulated datasets whose read lengths fall in the scope of 60bp ~ 150bp. As the length of reads grew from 50bp to 150bp, the recalls of all the assemblers improved. The improvements of IsoTree, Trinity, and BinPacker were particularly obvious. While the read length increased from 50bp to 150bp, the recalls of IsoTree, Trinity, and BinPacker improved from 0.41, 0.22, and 0.29 to 0.60, 0.47, and 0.44,
respectively. Because IDBA-Tran was designed especially for short reads, it could not work while the read length is larger than 130bp.

For the precision measure, IsoTree always held the highest precision among all the assemblers on all simulated samples (Figure 4b). The average precision held by IsoTree on these 21 samples was 0.62, which had 37.8%, 31.9%, 67.6%, 72.2%, 138.5%, 181.8%, 106.7%, and 93.8% increase over Trinity (0.45), BinPacker (0.47), SOAPdenovo-Trans (0.37), IDBA-Tran (0.36), Oases (0.26), Velvet (0.22), Cufflinks (0.30), and StringTie (0.32), respectively. The outstanding performance of IsoTree on precision measurement benefits from its special prune transcript method. IsoTree pruned the candidate transcripts by checking if the candidate transcripts are fully covered by paired-end mapped reads.

Considering that the length of k-mers plays an important role in de novo assemblers, we evaluated IsoTree, Trinity, BinPacker, SOAPdenovo-Trans, IDBA-Tran, Oases, and Velvet with different k values on the sample that contains 0.1 million 100bp paired-end reads (Table 1).

The experiments on different k values (Table 1) showed that the k value had a great influence on all the assemblers but IsoTree. This maybe result from IsoTree’s special extension strategy. IsoTree extends the contigs by reads directly while the other de novo assemblers mainly use k-mers. The read extension strategy can make full use of the information of nucleotides arrangement in reads. In most cases, the assemblers’ performance improved with the increase of k value. For example, Trinity, BinPacker, IDBA-Tran, SOAPdenovo-Trans, and Oases reached their best performances when k = 30. Velvet reached its best performance in the case of k = 25. It is interesting to find that IsoTree always held its superior position against all the other assemblers, even with their most favorite k values. The highest recalls of IsoTree, Trinity, BinPacker, IDBA-Tran, SOAPdenovo-Trans, Oases, and Velvet were 0.58, 0.50, 0.49, 0.29, 0.36, 0.37, and 0.21, respectively. The optimal precision of IsoTree was 0.66, which had 34.7% increase to Trinity’s highest precision (0.49), 6.5% increase to BinPacker’s best precision (0.62), 50.0% increase to IDBA-Tran’s optimal precision (0.44), 29.4% increase to SOAPdenovo-Trans’s best precision (0.51), 69.2% increase to Oases’s optimal precision (0.39), and 175.0% increase to Velvet’s highest precision (0.24).

For each assembler, we counted the number of full-length identified transcripts (that were produced under the optimal k-mer value) falling in each expression level region (Figure 5). We applied FPKM (the expected number of fragments per kilobase of transcript per million fragments sequenced) to measure the expression level of sequenced transcripts. We divided the transcripts abundance to 11 scopes: (0, 1000), (1000, 3000), (3000, 5000), (5000, 10000), (10000, 15000), (15000,20000), (20000, 25000), (25000, 30000), (30000, 35000), (35000, 40000] and (40000, max(transcript abundance)], and counted the number of full-length identified transcripts in each scope. Note that the length of reads involved in this experiment was 100bp.

![Fig. 5. Distribution of full-length identified transcripts that extracted by the assemblers. The horizontal axis shows transcript expression level (FPKM), and the vertical axis corresponds to assemble method. The area of each circle represents the number of full-length identified transcripts. The “Simulated” corresponds to the real number of sequenced transcripts falls in each expression scope in simulated experiments.](image)

From Figure 5, we observed that about a quarter of sequenced transcripts fell in the scope of 5000 ~ 10000, and Trinity obtained the most number of full-length identified transcripts in this scope, followed by IsoTree and SOAPdenovo-Trans. We found that both the low expressed transcripts and high expressed transcripts were difficult to

<table>
<thead>
<tr>
<th>Assembler</th>
<th>k= 15 Recall</th>
<th>k= 15 Precision</th>
<th>k= 20 Recall</th>
<th>k= 20 Precision</th>
<th>k= 25 Recall</th>
<th>k= 25 Precision</th>
<th>k= 30 Recall</th>
<th>k= 30 Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsoTree</td>
<td>0.58</td>
<td>0.64</td>
<td>0.58</td>
<td>0.66</td>
<td>0.57</td>
<td>0.65</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>Trinity</td>
<td>0.19</td>
<td>0.21</td>
<td>0.29</td>
<td>0.38</td>
<td>0.49</td>
<td>0.42</td>
<td>0.50</td>
<td>0.49</td>
</tr>
<tr>
<td>BinPacker</td>
<td>0.24</td>
<td>0.24</td>
<td>0.35</td>
<td>0.39</td>
<td>0.41</td>
<td>0.49</td>
<td>0.49</td>
<td>0.62</td>
</tr>
<tr>
<td>IDBA-Tran</td>
<td>0.17</td>
<td>0.22</td>
<td>0.20</td>
<td>0.27</td>
<td>0.27</td>
<td>0.39</td>
<td>0.29</td>
<td>0.44</td>
</tr>
<tr>
<td>SOAPdenovo-Trans</td>
<td>0.21</td>
<td>0.24</td>
<td>0.27</td>
<td>0.31</td>
<td>0.32</td>
<td>0.41</td>
<td>0.36</td>
<td>0.52</td>
</tr>
<tr>
<td>Oases</td>
<td>0.28</td>
<td>0.21</td>
<td>0.29</td>
<td>0.24</td>
<td>0.31</td>
<td>0.24</td>
<td>0.37</td>
<td>0.29</td>
</tr>
<tr>
<td>Velvet</td>
<td>0.11</td>
<td>0.12</td>
<td>0.12</td>
<td>0.14</td>
<td>0.21</td>
<td>0.24</td>
<td>0.20</td>
<td>0.24</td>
</tr>
</tbody>
</table>
recover. For example, there were 12 expressed transcripts falling in the scope of 0 ~ 1000, but only one transcript was recovered. For the scope of 35000 ~ 40000, there were total of four expressed transcripts, but no one was reconstructed. Surprisingly, IsoTree, Trinity, BinPacker, and IDBA-Tran reconstructed all the transcripts whose expression level fell in the scope of 25000 ~ 30000. We attribute the excellent performances of assemblers on this scope to the appropriate transcript expression level and to the specific sequence structure. It is worth mentioning that IsoTree recovered at least 50% of expressed transcripts in all the scopes except 0 ~ 1000 and 35000 ~ 40000. Trinity performed best when the transcript expression level fell in the scope of 5000 ~ 10000. BinPacker recovered the largest number of transcripts whose expression level fell in the scope of 15000 ~ 20000.

### 3.4 Real data

Instead of just running the de novo assemblers with all the default parameters, we tested the parameter of k-mer length with different values on the dog dataset and the human dataset, while keeping all the other parameters as default. We chose the best length of k-mer for each program, and compared their performances on both the dog dataset and the human dataset. We evaluated the de novo assemblers by the number of full-length reconstructed transcripts and the number of full-length identified transcripts on real datasets. We also examined their running times and memory usages on the dog dataset as a reference of their resources requirements.

We evaluated the performances of the assemblers with different lengths of k-mers by the number of full-length identified transcripts in Table 2.

On the dog dataset, most of the de novo assemblers achieved their best performances when the length of k-mer was 20bp. For two adjacent reads that were sequenced from the same transcript, the overlap length between them may be very long or short due to sequencing techniques. The k-mer-based extension strategy guarantees k - 1-length overlaps while our read-based extension strategy ensures at least k-length overlaps. The smaller k value, the higher probability that two adjacent reads can be connected. However, it will also lead to an increase in the number of error connection. As shown in Table 2, the number of candidate transcripts increased with the reduction of k-mer length. Trinity, BinPacker, and SOAPdenovo-Trans achieved their best performance with k = 20bp on the dog dataset, while their most suitable k values in the 100bp-length simulated sample were 30bp. We attribute this phenomenon to the differences in transcript expression levels between the dog dataset and the 100bp-length simulated dataset. As described in Introduction, longer k-mers tend to perform better on higher expressed transcripts, and shorter k-mers are more suitable for lower expressed transcripts. Further, in sequencing lower expressed transcripts, the overlap between two adjacent reads is usually shorter.

From Table 2, we drew the most suitable k-mer lengths for IsoTree, Trinity, BinPacker, IDBA-Tran, SOAPdenovo-Trans, Oases, and Velvet as 20bp, 20bp, 20bp, 30bp, 20bp, 25bp, and 30bp, respectively. In what follows, we evaluated these assemblers on the dog dataset under their most favorite k values.

With the most suitable lengths of k-mer, we compared the performances of these de novo assemblers by the number of transcripts with various identities as shown in Table 3. The 80%-length identified transcript is a reference transcript that is recovered by assemblers with at least 80% identity. The 80%-length reconstructed transcript is an assembled transcript that is at least 80% identical to a reference transcript. Identified transcripts and reconstructed transcripts with other identities are defined similarly.

On the dog dataset, IsoTree outperformed all the other de novo assemblers in terms of recovering both identified transcripts and reconstructed transcripts of all kinds of identities (Table 3). For example, IsoTree recovered 1354 full-length identified transcripts, a more than 17.8% increase over the number of full-length identified transcripts obtained by BinPacker (1149), Trinity (1017), SOAPdenovo-Trans (1005), IDBA-Tran (598), Oases (530), and Velvet (221). This means that IsoTree held the biggest recall among all these assemblers on the dog dataset. Besides, IsoTree not only assembled the most number of full-length reconstructed transcripts, and it also suggested less candidates than most of the other de novo assemblers. IsoTree, Trinity, BinPacker, IDBA-Tran, SOAPdenovo-Trans, Oases, and Velvet recovered 2974, 1663, 2601, 1011, 1006, 957, and 211 full-length reconstructed transcripts out of 81597, 96018, 73419, 69757, 85028, 113361, and 74530 candidates, respectively. From these numbers we can figure out that among all these assemblers, IsoTree is the most accurate. Except IsoTree, BinPacker outperformed all the other assemblers both with recall and

### Table 2: Comparison of de novo assemblers with different k values on dog dataset

<table>
<thead>
<tr>
<th>k-mer</th>
<th>Transcripts</th>
<th>IsoTree</th>
<th>Trinity</th>
<th>BinPacker</th>
<th>IDBA-Tran</th>
<th>SOAPdenovo-Trans</th>
<th>Oases</th>
<th>Velvet</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=20</td>
<td>full-length</td>
<td>1354</td>
<td>1017</td>
<td>1149</td>
<td>399</td>
<td>1005</td>
<td>512</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>candidate</td>
<td>81597</td>
<td>96018</td>
<td>73419</td>
<td>61429</td>
<td>85028</td>
<td>151856</td>
<td>36107</td>
</tr>
<tr>
<td>k=25</td>
<td>full-length</td>
<td>1158</td>
<td>992</td>
<td>1085</td>
<td>462</td>
<td>939</td>
<td>530</td>
<td>225</td>
</tr>
<tr>
<td></td>
<td>candidate</td>
<td>76067</td>
<td>84984</td>
<td>52956</td>
<td>67585</td>
<td>81681</td>
<td>113361</td>
<td>74530</td>
</tr>
<tr>
<td>k=30</td>
<td>full-length</td>
<td>867</td>
<td>808</td>
<td>990</td>
<td>598</td>
<td>800</td>
<td>416</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>candidate</td>
<td>65357</td>
<td>75081</td>
<td>46148</td>
<td>69757</td>
<td>85590</td>
<td>110179</td>
<td>71747</td>
</tr>
</tbody>
</table>

---

1545-5963 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
TABLE 3
Performance of de novo assemblers on dog dataset

<table>
<thead>
<tr>
<th>Assembler</th>
<th>full-length</th>
<th>90%-length</th>
<th>85%-length</th>
<th>80%-length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>identified</td>
<td>reconstructed</td>
<td>identified</td>
<td>reconstructed</td>
</tr>
<tr>
<td>IsoTree</td>
<td>1354</td>
<td>2974</td>
<td>1510</td>
<td>3357</td>
</tr>
<tr>
<td>Trinity</td>
<td>1017</td>
<td>1663</td>
<td>1147</td>
<td>1881</td>
</tr>
<tr>
<td>BinPacker</td>
<td>1149</td>
<td>2601</td>
<td>1304</td>
<td>2918</td>
</tr>
<tr>
<td>IDBA-Tran</td>
<td>598</td>
<td>1011</td>
<td>669</td>
<td>1135</td>
</tr>
<tr>
<td>SOAPdenovo-Trans</td>
<td>1005</td>
<td>1006</td>
<td>1120</td>
<td>1212</td>
</tr>
<tr>
<td>Oases</td>
<td>530</td>
<td>957</td>
<td>608</td>
<td>1137</td>
</tr>
<tr>
<td>Velvet</td>
<td>221</td>
<td>221</td>
<td>267</td>
<td>267</td>
</tr>
</tbody>
</table>

TABLE 4
Performance of de novo assemblers on human dataset

<table>
<thead>
<tr>
<th>Assembler</th>
<th>full-length</th>
<th>90%-length</th>
<th>85%-length</th>
<th>80%-length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>identified</td>
<td>reconstructed</td>
<td>identified</td>
<td>reconstructed</td>
</tr>
<tr>
<td>IsoTree</td>
<td>2015</td>
<td>3842</td>
<td>2422</td>
<td>4645</td>
</tr>
<tr>
<td>Trinity</td>
<td>1913</td>
<td>3039</td>
<td>2305</td>
<td>3850</td>
</tr>
<tr>
<td>BinPacker</td>
<td>1491</td>
<td>3449</td>
<td>1805</td>
<td>4234</td>
</tr>
<tr>
<td>IDBA-Tran</td>
<td>1376</td>
<td>2196</td>
<td>1666</td>
<td>2692</td>
</tr>
<tr>
<td>Oases</td>
<td>1762</td>
<td>3126</td>
<td>2139</td>
<td>3975</td>
</tr>
<tr>
<td>Velvet</td>
<td>151</td>
<td>151</td>
<td>194</td>
<td>194</td>
</tr>
</tbody>
</table>

precision since it reconstructed the second most (1149) full-length identified transcripts and reconstructed the second most full-length reconstructed transcripts with contrast to its candidates (2601 to 73419). IDBA-Tran was complementary to SOAPdenovo-Trans. IDBA-Tran obtained a larger number of full-length reconstructed transcripts, 90%-length reconstructed transcripts, 85%-length reconstructed transcripts, and 80%-length reconstructed transcripts than SOAPdenovo-Trans did (1011, 1135, 1267, and 1405 vs 1006, 1121, 1216, and 1312, respectively). However, the numbers of full-length identified transcripts, 90%-length identified transcripts, 85%-length identified transcripts, and 80%-length identified transcripts obtained by SOAPdenovo-Trans were obviously larger than the numbers of identified transcripts recovered by IDBA-Tran (1005, 1120, 1214, and 1309 vs 598, 669, 744, and 841, respectively).

We also run the above seven de novo assemblers on the human dataset with the above k-mer lengths, i.e., IsoTree with 20bp, Trinity with 20bp, BinPacker with 20bp, IDBA-Tran with 30bp, Oases with 25bp, and Velvet with 30bp, respectively. All software finished their works within a week except SOAPdenovo-Trans. Since SOAPdenovo-Trans was still in the state of reading reads from the second reads file after two weeks of running, we excluded it from our experiments on the human datasets. The numbers of reconstructed transcripts and identified transcripts with identities of 95% (full-length), 90%, 85%, and 80% are shown in Table 4.

On the human dataset, IsoTree maintained its superior performance on recovering identified transcripts with all kinds of identities (Table 4). For example, IsoTree reported 2015 full-length identified transcripts and improved Trinity (1913), BinPacker (1491), IDBA-Tran (1376), and Oases (1762) with 5.3%, 35.1%, 46.4%, and 14.4%, respectively.IsoTree, Trinity, BinPacker, IDBA-Tran, and Oases collected 3821, 3039, 3449, 2196, and 3126 full-length reconstructed transcripts out of 218269, 437730, 192674, 182651, and 439865 candidates, respectively. From these numbers we can figure out that IsoTree in general outperformed all the other assemblers. BinPacker is a litter more accurate than IsoTree since BinPacker assembled 3449 full-length reconstructed transcripts out of 192674 candidates while the numbers for IsoTree are 3821 to 218269. However, IsoTree constructed much more full-length identified transcripts than BinPacker, i.e., 2015 to 1491, which suggests a much higher recall. IsoTree outperformed all the other assemblers except BinPacker with both recall and precision. For the other identities, we can draw the same conclusion.

Assemblers usually behave differently on different datasets. For example, Trinity recovered more full-length identified transcripts and full-length reconstructed transcripts than BinPacker did on the human dataset. However, Trinity reported fewer full-length identified transcripts and full-length reconstructed transcripts than BinPacker did on the dog dataset. Oases collected more full-length identified transcripts from the human dataset, whereas BinPacker obtained more full-length identified transcripts than Oases.
on the dog dataset. Besides, Oases performed better on the human dataset than on the dog dataset.

![Graph](image)

Fig. 6. Computational requirements. (a) Peak memory for each de novo assembler on dog dataset. (b) Running time for each de novo assembler on dog dataset.

We also assessed the computational demands of these de novo assemblers with regard to running time and peak memory (Figure 6). There was an obvious phenomenon that the assemblers that performed outstandingly usually cost more time and memory, e.g., Trinity, IsoTree and BinPacker. Fortunately, the time and memory spent by these assemblers are acceptable. IsoTree took less time than Trinity and consumed almost the same amount of memory as BinPacker. Velvet cost the shortest time, while IDBA-Tran consumed the least memory.

### 4 Conclusion

We presented a novel de novo method IsoTree for transcriptome reconstruction from RNA-seq reads. We constructed the splicing graph by drawing the advantages of the state-of-the-art methods and adding our own consideration of expanding contigs by reads directly. We applied a mixed integer linear program model subtly to build the isoform tree which could express the potential transcripts in a gene. Since the computation is carried on one splicing graph after another, and the size of the splicing graph differs not much for different gene, IsoTree is scalable for most simulated and real data. In addition, the process of pruning transcripts with help of read information has greatly improved the precision. The experiments on simulated samples showed that IsoTree always held the highest recall and precision than the other assemblers. The experiments on real datasets also proved the outstanding performance of IsoTree. Except that on the human dataset BinPacker got a little bigger value is suitable for another, and the size of the splicing graph differs not much for different gene, IsoTree is scalable for most simulated and real data. In addition, the process of pruning transcripts with help of read information has greatly improved the precision. The experiments on simulated samples showed that IsoTree always held the highest recall and precision than the other assemblers. The experiments on real datasets also proved the outstanding performance of IsoTree. Except that on the human dataset BinPacker got a little bigger recall, IsoTree in general outperformed all the compared de novo assemblers on recovering the full-length reconstructed transcripts and the full-length identified transcripts with an acceptable computational demand. The evaluation of de novo assemblers with different \( k \)-mer lengths showed that the length of \( k \)-mers plays an important role on de novo assemblers’ performances. The small \( k \) value is suitable for lower expressed transcripts, whereas the high expressed transcripts favor longer \( k \)-mers. Besides, the large-scale of RNA-seq reads increased the difficulty of transcripts assembly. The advent of single-cell sequencing technology reduced the scale of the data. However, single-cell sequencing will result in uneven sequencing. Some fragments of a transcript may be sequenced many times, while some fragments of the transcript may not be sequenced at all [32]. A novel strategy specially designed for processing single-cell sequencing reads is required.

### Acknowledgments

This work is supported by National Natural Science Foundation of China under No. 61672325 and No. 61472222.

### References


