Sequence analysis

Informed kmer selection for de novo transcriptome assembly

Dilip A. Durai¹,² and Marcel H. Schulz¹,²,*

¹Cluster of Excellence on Multimodal Computing and Interaction, Saarland University, Saarbrücken, 66123, Germany and ²Department for Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarbrücken, 66123, Germany

*To whom correspondence should be addressed.

Associate Editor: Gunar Ratsch

Received on April 7, 2015; revised on December 31, 2015; accepted on April 17, 2016

Abstract

Motivation: De novo transcriptome assembly is an integral part for many RNA-seq workflows. Common applications include sequencing of non-model organisms, cancer or meta transcriptomes. Most de novo transcriptome assemblers use the de Bruijn graph (DBG) as the underlying data structure. The quality of the assemblies produced by such assemblers is highly influenced by the exact word length $k$. As such no single kmer value leads to optimal results. Instead, DBGs over different kmer values are built and the assemblies are merged to improve sensitivity. However, no studies have investigated thoroughly the problem of automatically learning at which kmer value to stop the assembly. Instead a suboptimal selection of kmer values is often used in practice.

Results: Here we investigate the contribution of a single kmer value in a multi-kmer based assembly approach. We find that a comparative clustering of related assemblies can be used to estimate the importance of an additional kmer assembly. Using a model fit based algorithm we predict the kmer value at which no further assemblies are necessary. Our approach is tested with different de novo assemblers for datasets with different coverage values and read lengths. Further, we suggest a simple post processing step that significantly improves the quality of multi-kmer assemblies.

Conclusion: We provide an automatic method for limiting the number of kmer values without a significant loss in assembly quality but with savings in assembly time. This is a step forward to making multi-kmer methods more reliable and easier to use.

Availability and Implementation: A general implementation of our approach can be found under: https://github.com/SchulzLab/KREATION.

Supplementary information: Supplementary data are available at Bioinformatics online.

Contact: mschulz@mmci.uni-saarland.de

1 Introduction

With the massive amounts of RNA-seq data (Sultan et al., 2008) produced for many non-model organisms, the interest for de novo analyses of RNA-seq data has increased over the last few years. These analyses include the de novo assembly of full length transcripts, expression level computation of novel transcripts, detecting differentially expressed transcripts and detection of related genes in close species or polymorphism detection (Davidson and Oshlack, 2014; Haznedaroglu et al., 2012; Le et al., 2013; Sloan et al., 2012).

Due to the large number of applications, several methods have been proposed for the de novo transcriptome assembly. Most of these methods are based on building a de Bruijn graph (DBG) from the RNA-seq reads. The nodes of the DBG are substring of length $k$ (also known as kmers) obtained from the reads and two nodes are connected if they have $k$-1 overlap. After obtaining the DBG, various heuristic algorithms are used to predict alternative transcripts from each DBG component. These algorithms either use a single kmer value (Chang et al., 2015; Grabherr et al., 2011; Xie et al.,...
2014) or merge transcripts generated from multiple kmer values to obtain a final non-redundant assembly (Peng et al., 2013; Robertson et al., 2010; Schulz et al., 2012; Surget-Groba and Montoya-Burgos, 2010). A multi-kmer based approach outperforms a single-kmer based approach as one kmer size rarely fits all genes (Chang et al., 2015; Peng et al., 2013; Schulz et al., 2012). Large kmer values resolve repeats and regions with many errors, whereas small kmer values are necessary to connect lowly expressed transcripts that have low-coverage regions. Hence even for most single kmer methods it is beneficial to run the assembler for different kmer values and merge the final results.

But the question about the core set of kmer sizes that are needed to achieve a good quality assembly has received little attention for multi-kmer methods. Currently, approaches (i) use the default kmer series of the assembler which is tuned on a limited number of datasets, (ii) select an arbitrary subset of kmer sizes or (iii) use all possible kmer sizes for the assembly. In practice that means that most often the resulting assembly is suboptimal. Either important kmer sizes are missed (cases 1–2) and therefore the sensitivity is reduced or the complete assembly contains more misassemblies than necessary and has wasted computational resources (case 3).

Selecting a suitable kmer size for de novo assembly has been previously explored in the context of genomic sequencing. For example (Chikhi and Medvedev, 2014) devised a method to select the best kmer size for de novo genome assembly. Simpson devised a method to estimate a number of interesting characteristics like paths with variations or repeats in the DBG for different kmer sizes from a FM-index over the reads (Simpson, 2014). Further, computing an optimal kmer range for de novo read error correction was also proposed (Ilie et al., 2011; Schulz et al., 2014). However, all these methods assume a uniform coverage distribution and are not applicable to non-uniform RNA-seq data.

Here we investigate in detail how the number of kmer assemblies can be minimized to save computational resources, without a large loss in sensitivity and without using a reference annotation for assembly quality assessment. We introduce the KREATION (Kmer Range Estimation) algorithm that is based on two novel contributions: (i) a comparative clustering of single kmer assemblies to define extended clusters which gives a notion of the assembly quality and (ii) a heuristic model assessment that allows to predict the optimal stopping point for a multi-kmer assembly method. We show that our new de novo strategy automates the choice of kmer sizes to explore, while achieving close to optimal performance.

2 Methods

2.1 Evaluation of assemblies

We use standard metrics for the evaluation of our assembled transcripts. We align transcripts against the reference genome using Blat (Kent, 2002, version 36) and compare it with annotated Ensemble transcripts (Cunningham et al., 2014, version 65). Then we compute the number of Ensembl transcripts that are overlapped to at least 80 or 100% by an assembled transcript, and term them as 80 and 100%-hits, respectively. We defined 80%-hit improvement rate as the ratio of the 80%-hits obtained from the multi-kmer assembly compared to a single-kmer based assembly. Further, we compute misassemblies by counting the number of aligned transcripts whose aligned region is ≤95% of the total transcript length. All transcripts that are not misassemblies are considered to be correct. We also compute nucleotide sensitivity and specificity as explained by (Schulz et al., 2012).

2.2 Transcriptome assembly

We used the de novo transcriptome assemblers Oases (Schulz et al., 2012, version 0.2.08), SOAPdenovo-Trans (Xie et al., 2014, version 1.03) and Trans-ABySS (Robertson et al., 2010 version 1.5.3) for our analyses. All assemblers were run with default parameters except the kmer parameter for the DBG and insert length for the paired-end sequencing data sets. Transcripts shorter than 100bps were removed from the final assembly.

SOAPdenovo-Trans doesn’t have its own merge script and the merge script of Oases and Trans-ABySS provides little information about the clusters obtained. Further, it was shown that the oases_merge script has suboptimal performance compared to CD-HIT-EST clustering (Hazaradaroglu et al., 2012). Hence we used CD-HIT-EST (Fu et al., 2012, version 4.6.1-2012-08-07) for clustering individual kmer assemblies, as detailed in Section 2.3.

In order to analyze how selecting arbitrary kmer values influences assembly performance we created three sets of random kmers. All the sets contained kmers spread across the read length, see Table 1.

We denote as best-k the assembly where the highest number 80%-hits was achieved for a dataset. For the analyzed datasets these were assemblies obtained from $k=25$ for brain, $k=23$ for human Embryonic Stem Cell (hESC), $k=27$ for T-cell and HeLa.

2.3 Clustering assembled transcripts

We use the CD-HIT-EST software, for clustering transcripts assembled by individual kmer sizes and merging sets of transcripts from distinct kmers. CD-HIT-EST is a fast clustering technique that reports final clusters with all sequences contained in the cluster. It uses a greedy algorithm to iteratively grow clusters and multicore parallelization for fast clustering. We run CD-HIT-EST with 10 threads.

Consider two sets of assembled transcripts, $T_1 = \{t_1, \ldots, t_n\}$ and $T_2 = \{t_1, \ldots, t_m\}$ with $n$ and $m$ many sequences respectively, each produced by running a de novo assembler for one or more kmer values. Let $C = \{c_1, \ldots, c_l\}$ be the set of $l$ sequence clusters obtained by applying a sequence clustering algorithm to $T_1$ and $T_2$, denoted as ComputeClusters($T_1, T_2$). We define the following notions on clusters: a cluster $c \in C$ is called unique with respect to $T_2$ if it holds that $c \subset T_2$, namely that all sequences in $c$ are only from assembly $T_2$. Further we call the longest sequence of a cluster $c \in C$ the representative of $c$, denoted as rep($c$). A cluster $c \in C$ is called extended by $T_2$ if rep($c$) $\in T_2$ and $c$ is not unique. We denote $c = \text{extended}(C, T_2)$ as the number of extended clusters in $C$ with respect to $T_2$ (See Supplementary Figure S1). Collectively we denote all clusters that are extended or unique with respect to $T_2$ as novel clusters with respect to $T_2$. All remaining cluster are called old.

2.4 Predicting the assembly stopping point via assessment of model fit

We analyzed assemblies generated by different de novo assemblers and observed that with an increase in kmer size the number of

| Table 1. Sets of random kmer values used to analyze the effect of selecting arbitrary kmer values on the assembly. Rows represents the sets and columns represent the dataset with read length mentioned in brackets |
|----------------|----------------|----------------|----------------|----------------|
| Brain (50) | TCell (45) | hESC (50) | HeLa (76) |
| Set1 | 23,33,37,45 | 25,33,37,45 | 25,33,37,45 | 25,39,53,61 |
| Set2 | 25,31,35,43 | 25,31,35,43 | 25,31,35,43 | 35,43,57,69 |
| Set3 | 23,33,37,45 | 23,33,37,45 | 23,33,37,45 | 51,55,69,71 |
correct transcripts produced by the assembler follows approximately an exponential distribution, see Results. We show that the number of extended clusters in consecutive assemblies behave similarly and can thus be used in a de novo setup. Our rationale was that once this exponential trend does not hold anymore for increasing kmer values, the number of extended clusters is not dominated by correct transcripts, but rather by misassemblies or redundant assemblies.

We summarize our approach for the above insight in algorithm 1: Given a set of reads and a minimal kmer, denoted \( k_{\text{min}} \), the assembler explores an \textit{a priori} fixed series of kmer values \( X = (k_1, \ldots, k_b) \) where \( k_1 = k_{\text{min}} \). This series is computed using a function \( f \). In our case we use the simple function: \( f(k) = k+2 \), as Oases and SOAPdenovo-Trans can use odd kmer values only and we wanted to use the same set of kmer values for all assemblers to maintain consistency. For each \( k \) an assembly is produced (line 7). All new transcripts in \( T_k \) are clustered with the previous transcripts using the function \( \text{ComputeClusters}(T, T_k) \) to produce \( C \) (line 8). The number of extended clusters \( e_i \) with respect to \( T_k \) is computed (using \( \text{extended}(C, T_k) \)) and its log count is stored in \( y_i \) (line 9).

We then assume the following linear model:

\[
Y = \beta_0 + \beta_1 X + \epsilon, \tag{1}
\]

where \( Y = (y_1, y_2, \ldots, y_n) \) denotes the series, with \( y_i = \log_{10}(e_i) \), \( \beta_0 \) and \( \beta_1 \) are the constants representing \textit{slope} and \textit{intercept} of the line respectively. Gaussian noise is denoted by \( \epsilon \).

We are interested in analyzing the linear model fit with an increase in number of data points. More precisely, if we have a linear model fit on \( n-1 \) datapoints (where \( n > 3 \), since we require at least three datapoints to model a line to avoid premature stopping of the assembly), we want to assess the error in predicting the \( n \)th datapoint using the line. We fit a line (lm) with \( Y = (y_1, y_2, \ldots, y_{n-1}) \) and \( X = (k_1, k_2, \ldots, k_{n-1}) \) and estimate the coefficients \( \tilde{\beta}_{n-1} = (\hat{\beta}_0, \hat{\beta}_1) \) by minimizing the residual sum of squares (line 10):

\[
\tilde{\beta}_{n-1} = \arg \min_\beta \sum_{i=1}^{n-1} (y_i - \hat{y}_i)^2, \tag{2}
\]

where \( \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 k_i \) is the predicted value of \( y_i \). Assuming that this line would also explain \( y_n \) we use the estimated coefficients to compute \( \hat{y}_n \) for \( k_n \) (line 11). We compute the error between \( y_n \) and its actual value:

\[
\text{err}(n) = (y_n - \hat{y}_n)^2. \tag{3}
\]

We increment \( n \) and repeat the above procedure until \( n \) reaches the stopping point. For any given kmer series of length \( m \), we define the \textit{deviation score} (\( d_{\text{score}} \)) as the cumulative sum of point-wise error estimates:

\[
d_{\text{score}} = \sum_{n=1}^{m} \text{err}(n). \tag{4}
\]

The stopping point is determined by applying a threshold to the \( d_{\text{score}} \) (line 13). The \( d_{\text{score}} \) is expected to remain close to zero for data points which follow a linear trend. It increases considerably and crosses the threshold value at a point where the quality of the linear fit degrades i.e. the fitted line is no longer able to explain the additional datapoint. If the \( d_{\text{score}} \) is less than the cutoff, the algorithm continues and updates all the variables (line 16–18). Otherwise the algorithm terminates and produces the final assembly.

### 2.5 De novo removal of misassemblies

A disadvantage of merging several single kmer assemblies is the increased number of misassemblies in the final result. In principle, misassembled transcripts should only occur at kmer values that are shorter than repeat length and thus they are unlikely to occur at all different kmer values. To accommodate this idea we devised the following method. Assume we run our assembly for the values \( k = (k_1, k_2, k_3, k_4) \). After producing the final clustering \( C \) of the transcripts of these four assemblies \( T_{k_1}, T_{k_2}, T_{k_3}, T_{k_4} \) we consider the clusters \( C \) in which only transcripts of a certain \( T_{k_i} \) exist, i.e. which are unique with respect to \( T_{k_i} \). We termed these clusters as \textit{single-k} clusters. We classified all the single-k clusters and transcripts shorter than a predefined length threshold (300 bp for all datasets) as misassemblies.

For measuring the difference in misassemblies between multi-kmer assembly (say \( T_{\text{multi}} \)) and single-kmer assembly (say \( T_{\text{single}} \)) we define:

\[
\text{misassembly rate} = \frac{\text{observed}}{\text{expected}} \tag{5}
\]

where,

\[
\text{observed} = \frac{\#\text{misassemblies in } T_{\text{multi}}}{|T_{\text{multi}}|}, \tag{6}
\]

\[
\text{expected} = \frac{\#\text{misassemblies in } T_{\text{single}}}{|T_{\text{single}}|}. \tag{7}
\]

### 2.6 Data retrieval and preprocessing

All datasets were downloaded from the SRA (http://www.ncbi.nlm.nih.gov/sra). Five RNA-seq datasets were used for analysis: 147M paired-end reads of length 50 bps for human brain (Barbosa-Morais...
of kmers, which by definition is optimal, choosing a random set or picking the Best-\(k\) shows a loss in performance up to 30\%. In particular, each setup outperforms the others on a different dataset, such that no one strategy can be recommended. KREATION achieves close to optimal performance (Fig. 1).

3.2 Clustering of consecutive kmer assemblies reveals assembly progress

The main focus of this work is to define a data-driven procedure that predicts an optimal stopping point for a de novo multi-kmer assembler. As we mentioned earlier, we want to find a subset \(\hat{K}\) that shows a similar number of 80\%-hits compared to the full assembly. First, we investigated the contribution of each single-kmer assembly to the performance of the full assembly using Oases. In Figure 2, we plot the number of correct transcripts that are assembled in their longest form in a particular single kmer assembly (x-axis). It can be seen that the assembly with the smallest kmer value contributes most of the correct transcripts and that a decreasing exponential trend can be observed, with higher kmer values being less important. From this we conclude that the problem of finding \(\hat{K}\) from all possible subsets \(\sum_{i=1}^{n} \binom{n}{i}\) can be simplified. We suggest to consider a series of increasing kmer values \((k_1, \ldots, k_n)\), where \(k_1 < k_2, \ldots, < k_n\). In this work we used the series of kmers \(k_i = k_{i-1} + 2\).

In Figure 3 (bottom) it is shown how the performance (% of optimal, left y-axis) changes when the multi-k assembly is constructed up to index \(i\) in the series, i.e. \(T_i = T_{k_1} \cup \ldots \cup T_{k_i}\), by merging transcripts, (see ‘Methods’). Similar to our observation in Figure 2 we see that higher \(k\) values contribute little to the assembly, e.g. 99\% of the optimal sensitivity is reached at \(k = 39\).

In a de novo circumstance, how can we predict the index \(i\) such that the obtained sensitivity is close to optimal sensitivity? We observed that the number of transcripts going from \(t\) to \(i + 1\) always increases, i.e. \(|T_i| > |T_{i+1}|\), illustrated for a dataset in Supplementary Table 1. However, we know from Figure 2 that for higher \(k\) values the number of correct transcripts decreases approximately exponentially. Correct transcripts, as in Figure 2, need to be novel. These novel transcripts in \(T_{i+1}\) are the representative sequences of clusters that either extend transcripts in \(T_i\) (extended clusters) or form unique clusters, without any transcript from \(T_i\).
For KREATION we use the CD-HIT-EST algorithm that preserves sequence to cluster memberships in its output and its clustering heuristic has an added advantage of being very fast in practice.

In our *de novo* setup some of the novel clusters will represent misassemblies. These misassemblies generally arise from the unique clusters (Supplementary Figure S2). Extended clusters constitute more 80%-hits than unique clusters (Supplementary Figure S3). Therefore, we can approximate the number of correct transcripts for $T_{i+1}$ by counting extended clusters.

In Figure 3 (top) we show how the logarithm of the number of extended clusters for the brain dataset behaves for different $k$ values. As the number of extended clusters also contains misassemblies we observed the exponential trend, visible as a line in log space, only up to $k = 35$. Afterwards, when fewer genuine transcripts are contributed by $T_{i+1}$, this trend changes. Hence, once the curve starts to deviate significantly from a straight line, dropping towards small cluster numbers, it may be advisable to stop the assembly as we do not expect contributions to the overall assembly.

### 3.3 Stopping the assembly by assessment of model fit

But how to measure if the exponential fit is worse after adding another $k$mer assembly?

We tested the following method. Suppose we run our assembly for values $k = k_1, k_2, \ldots, k_n$. We first fit a linear model to the log counts of extended clusters for all indices until $n - 1$, (see 'Methods'). Assuming that the predicted linear model would also explain the response variable of the current index $n$, we predict the log count of extended clusters for $n$. We then deduce the error between the actual value and the predicted value. The process is repeated for all values of $n$ and after each iteration errors are summed up defining our deviation score ($d_{score}$). Normally the error is close to zero for data points which follow a linear trend and hence the resulting $d_{score}$ is very small. The error becomes significantly larger at a point where the curve starts to deviate from the straight line. This results in a noticeable increase in $d_{score}$. As seen in Figure 3 (top), the value of $d_{score}$ remains close to zero until $k = 37$. Namely, until a $k$mer value where all the points are well approximated by a straight line. Beyond this $k$mer the $d_{score}$ increases significantly and goes above a pre-defined threshold (0.01 in our case). This point corresponds to a $k$mer that shows close to 99% of the optimal value (Fig. 3 bottom, dotted line). Therefore, stopping at this point results in almost no loss of assembly sensitivity and a significant reduction in runtime (Fig. 3 bottom, secondary axis). We tested this heuristics with other assemblers on the same dataset and found a similar behaviour (Fig. 4). Hence we used this heuristic to design the KREATION algorithm, see Algorithm 1.
Table 2. Analysis of assembly results with all assemblers on the five datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>kmer range</th>
<th>No. Transcripts</th>
<th>Sens. (%)</th>
<th>Spec. (%)</th>
<th>Oases</th>
<th>100%-hits</th>
<th>80%-hits</th>
<th>Runs saved</th>
<th>%hrs reduced</th>
<th>Runtime (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain KREATION</td>
<td>462 896</td>
<td>46.59</td>
<td>68.98</td>
<td>6264</td>
<td>42 540</td>
<td>6</td>
<td>32</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brain Full</td>
<td>468 056</td>
<td>46.74</td>
<td>68.94</td>
<td>6315</td>
<td>42 629</td>
<td>—</td>
<td>—</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hESC KREATION</td>
<td>196 824</td>
<td>37.58</td>
<td>73.13</td>
<td>12 783</td>
<td>42 899</td>
<td>8</td>
<td>31</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hESC Full</td>
<td>203 042</td>
<td>37.43</td>
<td>73.62</td>
<td>13 105</td>
<td>43 635</td>
<td>—</td>
<td>—</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HeLa KREATION</td>
<td>113 009</td>
<td>24.06</td>
<td>78.81</td>
<td>4243</td>
<td>25 625</td>
<td>15</td>
<td>10</td>
<td>162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HeLa Full</td>
<td>119 009</td>
<td>24.1</td>
<td>77.43</td>
<td>4284</td>
<td>25 721</td>
<td>—</td>
<td>—</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCell KREATION</td>
<td>129 868</td>
<td>19.11</td>
<td>77.91</td>
<td>3050</td>
<td>18 650</td>
<td>5</td>
<td>15</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCell Full</td>
<td>132 975</td>
<td>19.05</td>
<td>75.42</td>
<td>3131</td>
<td>18 918</td>
<td>—</td>
<td>—</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMR90 KREATION</td>
<td>1 362 744</td>
<td>49.65</td>
<td>37.21</td>
<td>22 165</td>
<td>59 253</td>
<td>14</td>
<td>34</td>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMR90 Full</td>
<td>1 384 061</td>
<td>49.55</td>
<td>36.75</td>
<td>22 246</td>
<td>59 700</td>
<td>—</td>
<td>—</td>
<td>901</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dataset kmer range No. Transcripts Sens. (%) Spec. (%) 100%-hits 80%-hits Runs saved %hrs reduced Runtime (hrs)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>kmer range</th>
<th>No. Transcripts</th>
<th>Sens. (%)</th>
<th>Spec. (%)</th>
<th>100%-hits</th>
<th>80%-hits</th>
<th>Runs saved</th>
<th>%hrs reduced</th>
<th>Runtime (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brain KREATION</td>
<td>295 870</td>
<td>33.28</td>
<td>63.08</td>
<td>4192</td>
<td>25 416</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Brain Full</td>
<td>298 286</td>
<td>33.28</td>
<td>63.35</td>
<td>4235</td>
<td>25 621</td>
<td>—</td>
<td>—</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>hESC KREATION</td>
<td>226 974</td>
<td>35.62</td>
<td>60.34</td>
<td>8072</td>
<td>34 914</td>
<td>5</td>
<td>25</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>hESC Full</td>
<td>229 635</td>
<td>35.45</td>
<td>60.27</td>
<td>8128</td>
<td>35 087</td>
<td>—</td>
<td>—</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>HeLa KREATION</td>
<td>139 955</td>
<td>25.63</td>
<td>61.89</td>
<td>3413</td>
<td>23 308</td>
<td>15</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>HeLa Full</td>
<td>144 909</td>
<td>25.63</td>
<td>60.3</td>
<td>3456</td>
<td>23 559</td>
<td>—</td>
<td>—</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>TCell KREATION</td>
<td>94 233</td>
<td>18.79</td>
<td>73.37</td>
<td>2598</td>
<td>16 589</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>TCell Full</td>
<td>94 925</td>
<td>18.8</td>
<td>73.48</td>
<td>2624</td>
<td>16 634</td>
<td>—</td>
<td>—</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>IMR90 KREATION</td>
<td>1 836 920</td>
<td>38.58</td>
<td>8.65</td>
<td>11 936</td>
<td>45 276</td>
<td>17</td>
<td>10</td>
<td>573</td>
<td></td>
</tr>
<tr>
<td>IMR90 Full</td>
<td>1 847 582</td>
<td>38.59</td>
<td>9.71</td>
<td>12 720</td>
<td>47 308</td>
<td>—</td>
<td>—</td>
<td>635</td>
<td></td>
</tr>
</tbody>
</table>

The first row for each dataset analyses the transcripts obtained by running the assembly till the optimal kmer. The second row for each dataset represents the results obtained by running the assembly till a feasible kmer closest to the read length. KREATION used a d_score threshold of 0.01 for all the cases.

3.3.1 Application to other datasets
In addition to the brain dataset, we tested KREATION on four other data sets with different read length and coverage (see ‘Methods’). In all cases the point where d_score crosses the threshold value coincides with a kmer value close to the optimum (see Supplementary Figure S4–S7).

We list the complete numbers for all five datasets in Table 2. As the table shows, consistently for all datasets the stopping point only leads to a small decrease in the 100 and 80%-hits. Further the final number of transcripts is reduced without affecting the nucleotide sensitivity and specificity. We also show the number of kmer assemblies not computed and the time saved due to KREATION. In particular, for the longer read datasets (HeLa and IMR90) KREATION avoids a significant number of assemblies, saving up to days of computation on our computing cluster. Note that the clustering done after each kmer assembly, only takes in the order of a few minutes as CD-HIT-EST is parallelized and very fast (Fu et al., 2012).

We found that selecting a threshold of 0.001 leads to at least a 50% reduction in runtime, and at the same time a loss up to at most 9% in comparison to the full assembly. A high threshold value results in achieving an almost 100% optimal assembly, but results in insignificant reduction of runtime (Fig. 5). For our analyses we chose a threshold value of 0.01, which seems to be a good tradeoff between runtime and quality of the final assembly, but other values may be preferred by the user.

3.4 Single-k clusters are enriched in misassemblies
The major drawback of a multi-kmer based assembly is the generation of a high number of misassemblies. We observed, and also various studies have shown, that misassemblies are generally shorter in length (see Supplementary Figure S8) and hence a large percentage of them can be removed by applying a length cutoff (300 bp in our case) on the final transcripts (Fig. 6 top). As clustering transcripts produced by different kmer assemblies is an integral part of KREATION, we wondered how misassemblies are distributed over the clusters. We observed that clusters which contained transcripts from only one kmer value consisted of a large number of misassemblies. We termed these clusters single-k clusters (see ‘Methods’). Removing these clusters also reduces the number of misassemblies.
significantly (Fig. 6 top). We found that removing all transcripts which are either shorter than the length cutoff or which belong to single-k clusters reduces more misassemblies as compared to applying only one of the above mentioned filters (Fig. 6 top).

Further we wanted to check whether merging assemblies from multiple kmer values as done in KREATION is better than the single-kmer assembly with the highest sensitivity ($k_1$ in our case). In other words, we wanted to check whether we are generating more misassemblies than useful transcripts, when we merge assemblies from multiple kmer values. For this, we define 80%-hit improvement rate as the fold-change of 80%-hits from the assembly generated by the lowest kmer and misassembly rate as the ratio of observed to the expected number of misassemblies. Figure 6 (bottom) shows the comparison between these two metrics for all KREATION assemblies computed, separated by dataset. In an ideal situation, the 80%-hit improvement rate should be better than the misassembly rate and hence all the points in the graph should be above the diagonal. We show that for most of the datasets the corresponding points are either above the diagonal or close to the diagonal, except for the HeLa dataset assembled by TransABYSS, which has a misassembly rate of 1.93. We conclude that the multi-kmer based assembly approach followed by appropriate filtering of misassemblies is better than the single kmer based assembly.

4 Discussion and conclusion

We have presented KREATION, an algorithm that is able to automatically stop a de novo multi-k transcriptome assembly at a kmer value close to the optimal sensitivity. We showed that clustering newly assembled transcripts with all assemblies made in previous rounds can be used to estimate how many transcripts have been improved due to the last round, by counting extended clusters. We empirically found that the number of extended clusters falls exponentially with increasing k, when we consider to run the multi-k as assembler from smaller kmer values to larger ones, as is most reasonable for transcriptome assemblies. This motivated us to predict the kmer value at which the assembly should stop by selecting the kmer where deviation to the expected exponential trend deviates considerably, as measured with the designed the d_score. We have shown that thresholding the d_score works well for three popular de novo assemblers, and datasets with different coverages and read lengths. For longer read datasets KREATION may save days of computation.
However, KREATION is a heuristic approach and there is no guarantee that the $d_{\text{score}}$ threshold corresponds to a good stopping point or that there will be an optimum for the set of kmers tested, albeit both is true for the datasets and assemblers tested in this work. Still, we think that there is a theoretical connection that is worth exploring further. KREATION often selects different stopping kmer values for each assembler when applied to the same dataset. This points to a complex interplay between graph structure, transcriptome complexity, read coverage and assembler implementations. It may be the reason why simpler strategies for selecting kmers do not seem to generalize for datasets with different characteristics (cf. Fig. 1).

One disadvantage of merging the assemblies of several $k$ values is the increased number of misassemblies as compared to using one kmer. However, recent studies have shown that appropriate filters (Yang and Smith, 2013) or a proper statistical treatment with replicates (Davidson and Oshlack, 2014) allows to remove most misassemblies and therefore ease downstream analyses. Here we show that a large number of misassemblies stem from single-k clusters that can be removed easily. Together with a length cutoff on assembled transcripts this denotes a powerful approach to get rid of most of the misassemblies without removing genuine transcripts (Fig. 6, Supplementary Figure S9). We argue that this makes multi-kmer assembly strategies more useful for the community, but we think that there is still room for improvement for removing misassemblies.

In a recent work (Li et al., 2014) a reference free transcriptome assembly evaluation approach was introduced. Based on a graphical model of the RNA-sequencing process, an assembly quality estimate can be computed by aligning reads to the assembled transcripts. We note that in principle their method may be used as an alternative function to decide when to stop. However, this approach would constitute a serious runtime cost as read alignment, SAM file writing and model building would take in the order of hours for one kmer iteration. Instead, we see their work complementary to our work. We have focused on the question where to stop the assembly, but there are other parameters that are worth tuning. For example, we are currently assuming the $k_i$ is given, which is partly due to the fact that it just needs to be chosen in such a way to avoid small kmers that produce misassemblies.

As a conclusion, we show that an informed kmer selection approach for de novo transcriptome assembly shows an improvement over simpler methods suggested so far. We believe that KREATION with the misassembly filters will be useful for the community and implemented the mentioned ideas in a software that currently supports the de novo assemblers tested in this work (https://github.com/SchulzLab/KREATION).

## Acknowledgement

We like to thank the anonymous reviewers for their constructive comments.

## Funding

This work was supported by the Cluster of Excellence on Multimodal Computing and Interaction (EXC284) of the German National Science Foundation (DFG).

## References