A machine learning pipeline to improve De Bruijn graph metatranscriptomic assemblies

Hussein Mohsen
School of Informatics and Computing, Indiana University, Bloomington, IN 47405, USA

ABSTRACT

Motivation: With the growing significance of metatranscriptomic assemblies, the need to improve their quality and maintain their controllable size has become essential. That would help in boosting all applications based on metatranscriptomic assembly. In this paper, we propose a pipeline that filters de novo assemblies while preserving or improving their quality. Original assemblies are based on De Bruijn graphs and were created by Oases. Auxiliary scripts that help reporting statistics about all kinds of metatranscriptomic assemblies are integrated with the pipeline as well.

Results: Experimental results show that the pipeline helped improving the accuracy of the assemblies with up to 6+-% in addition to filtering 5000+ transcripts from 6 original assemblies each made up of 21000+ transcripts. The high precision of filtered assemblies and the reasonable running time of the pipeline makes it a potential postprocessing step of different de novo assemblies.

Availability: All pipeline scripts are publicly available at https://sourceforge.net/projects/metatranspipeline/files/

Contact: hmohsen@indiana.edu

1 INTRODUCTION

The analysis of multiple transcriptomes helps to reveal many secrets within microbial communities [6]. A transcriptome is the set of all RNA molecules in a cell. This includes the tRNA, rRNA, nRNA, and non-coding RNA present at the moment the sample is taken. Different sequencing technologies have been built to be applied on RNA sequences benefiting from the abundance of tools used for high-throughput DNA sequencing at the base level. RNA-Seq is one of the nucleotide-level technologies that deploys Next-Generation Sequencing (NGS) to help fulfilling this goal [2] [24], and it was first referred to in 2008 [13] [4] [15]. It allows for looking at post-transcriptional changes, gene fusion, alternative gene splicing events in eukaryotes, gene expression levels and mutations/SNPs [7]. In addition, and along with miRNA populations, RNA-Seq is applicable on varying RNA populations including small RNA, total RNA, ribosomal RNA and miRNA [12].

RNA-seq attains a significantly better coverage factor and precision of transcripts levels (and their isoforms) than those by microarrays. Specificity and sensitivity values proved to be better as lowly expressed and closely related sequences are better identified [17]. However, being a complex and cost-demanding technology makes RNA-seq less favorable than microarrays in different applications.

In metatranscriptomics, different methods are used to study community-wide samples usually made up of a vast number of transcripts that originate from multiple organisms. In nature, unlike in laboratories, bacteria grow together in complex communities and often with archaeal organisms. This makes studying natural metasamples more reflective of biological truth. Moreover, after the success of metagenomics in revealing important functions in bacterial communities, metatranscriptomics can provide complementary knowledge for further related applications on these communities. Meta-samples analysis often starts with sequence assembly. A number of de novo assemblers were implemented to address this problem. In this project, we build a machine learning pipeline that filters and hence improves de novo metatranscriptomic assemblies based on De Bruijn graphs.

2 MOTIVATION

Different solutions are available to overcome the obstacles encountered by reference-based assemblers, such as the unavailability of reference genomes or inability to identify targeted species. State-of-the-art de novo assemblers [21] [1] [18] [22] work in this direction but the quality of their output assemblies is still not close to that of their reference-based counterparts. However, the efficient execution of these (de novo) assemblers along with other factors make their utilization promising. With the advances in machine learning, we attempt to apply a sequence of filtering methods on de novo assemblies to refine their quality. These methods are ordered in a pipeline on which this project is centered. The main objective of this pipeline is to approach the quality of reference-based assemblies while benefiting from the advantages of de novo assembling strategies. That should be achieved after postprocessing original de novo assemblies produced Oases or any other De-Brujin-based assembler.

In formal set notation, the problems is describes as follows: let \( R \) be a reference-based assembly, \( D \) a de novo assembly and \( DP \) a de novo assembly refined by the pipeline. Let \( B = R \cap D \), \( A = R - D \) and \( C = D - R \). An intersection between two sets is characterized by the generation of reliable and similar transcripts. The aim of the pipeline is to increase the size of \( B \) and consequently decrease the sizes of \( A \) and \( C \). This can be expressed as \( B' = R \cap DP > B \cap A' = R - DP < A \) and \( C' = DP - R < C \). That is pictorially expressed by the Venn diagrams of Figure 1.

3 RELATED WORK

Different de novo approaches are adopted for assembling whole transcriptome using RNA-seq data. Three of the well-known assemblers - among which is Velvet-based [26] Oases- are based on De
Bruijn graphs. Trinity [18] proved to have an exemplary performance against eukaryotic samples, yet suffered from the problem of generating a small number of complete transcripts compared to that of fragments. Another assembler called ABySS [22] is known to have the fastest performance, yet low quality assemblies. TransAbyss [9], which is a pipeline based on ABySS, had a better performance but on human RNA-seq samples and not metatranscriptomic bacterial ones studied in this paper. MIRA [1] is another assembler based on a different kind of graph called the overlap graph. MIRA produces assemblies with a good quality yet suffers from being slow. That would be a more considerable burden in metatranscriptomic experiments than in ones that involve a single transcriptome.

The EM algorithm and other machine learning techniques have previously been used in applications different from de novo assembly filtering. In [3], this algorithm is used along with probabilistic methods to correct the expression levels of RNA-seq data and eliminate the noise. That would be helpful before running Oases and the pipeline described in this paper, and they all can be combined to form a larger pipeline if their total execution time is acceptable as per the time limits of the task under consideration. In [5], an EM algorithm is used to estimate gene expression values of unreliable or multiple mappings of reads. The algorithm is used in the context of a larger generative model based on likelihood probabilities. A significant use of the EM algorithm is embedded in RSEM [19], which is a de novo software package that works toward improving transcript quantification from RNA-seq data. One clear feature RSEM shares with our pipeline is having a de novo metatranscriptomic assembly as input. However, the objective of RSEM is different and is centered on improving the quantification of the transcripts rather than filtering them. Another pipeline called Rnnotator [11] works on a sample of reads and runs on the top of Velvet aiming for the removal of errors within transcripts. Rnnotator proved to have a good performance in purifying a considerable part of assembly transcripts but maintained all of them without filtering any.

Other systems are based on algorithms different from the EM and try to detect isoforms and splicing junctions, facilitate gene annotation, and quantify expression levels accordingly. The most famous among these are TopHat [23] and Cufflinks [8]. These systems among others try to remove the noise associated with read mappings and calculate better expression values. However, none of them use machine learning techniques to filter de novo assemblies and exclude unreliable transcripts as our pipeline does.

4 METHODS

Oases is based on a genome sequence assembler called Velvet. It arranges the nodes De Bruijn graph into clusters of overlapping nodes called Loci. Ideally, each locus should include nodes whose sequences correspond to the same gene. Transcripts are then generated within each locus. All generated transcripts of a certain locus are made up of one or more nodes from that same locus. A number of loci is simple and end up generating one transcript only. Other loci generate multiple transcripts. We call the first type of loci oneTransLoci and the latter multiTransLoci throughout the paper. The number of generated transcripts depends on the complexity of the structure of the locus’ subgraph and the dynamic programming algorithm used by Velvet. Accordingly, we divide our pipeline into two modules: one that filters multiTransLoci and another that filters oneTransLoci.

The first module that works on multiTransLoci passes through 3 phases as shown in the diagram of Fig. 2. It begins with an EM algorithm with a logistic sigmoidal basis function δ and ends with Oases transcripts filtering based on hamming distance passing through an intermediate step of transcript prediction. In the first phase, the EM algorithm takes as input the number of transcripts to remain from a locus after filtration. We call this number F and it is bounded by the total number of transcripts produced by the locus. Then, the algorithm tries to iteratively approximate a collective score for each of the F transcripts and maintains these scores in a list θ made up of F cells. Before describing the steps of the algorithm, an important definition of collective and individual scores of transcripts and nodes are described respectively. A collective score of a transcript depends on the custom individual score of each of its nodes. This individual score is described in equation (1) and considers three main factors: edge multiplicity, coverage, and length.

Each edge in the De Bruijn graph built by Oases has a multiplicity value greater than or equal to 1. This value indicates how many times an edge has been repeated while building the graph. Consequently, the higher the multiplicity value, the more reliable the edge that connects two k-mer nodes in the graph is. The average multiplicity of a certain node with its neighborhood is calculated and added to the coverage of this node. Similarly, the higher the coverage of a node, the more reliable it is. The total value is then multiplied by the weight of this node’s length compared to other nodes’ lengths.

**Fig. 1.** Venn diagrams of Ref-based, De novo and Pipeline assemblies
within the transcript. If a node belongs to more than one transcripts, its weight (based on its length) may vary depending on the lengths of other nodes within each transcript. The longer the node is, the more weight it is given.

\[
\text{score}(n) = \frac{\left( \left( \sum_{n \in E_n} \frac{n(e)}{|E_n|} \right) + \text{Cos}(n) \right) \cdot l(n)}{\sum_{n' \in N} l(n')}
\]

(1)

**Algorithm 1 EM algorithm**

1: procedure EM

   INPUT number of filtered transcripts \( F \),
   node scores \( S \), set of nodes \( N \)

   OUTPUT Collective transcript scores \( \theta \)

2: initialize \( \theta[F] \)

3: initialize \( \theta[[N][|F|]] \) % likelihood matrix %

4: \( \text{MAXITER} \leftarrow 25 \)

5: \( i \leftarrow 1 \)

6: while \( i \leq \text{MAXITER} \) do

7: \( \text{STIR MATRIX}() \)

8: for node in \( N \) do

9:   for \( \text{trans} \leftarrow 1 \) to \( F \) do

10:      \( \theta[\text{trans}] \leftarrow \delta(S[\text{node}] - \theta[\text{trans}]) \)

11: end for

12: end for

13: \( \text{nodeCount} \leftarrow 0 \)

14: for node in \( N \) do

15:   \( \text{maxTrans} \leftarrow \max(\theta[\text{node}][\text{trans}]) \)

16:   if \( \text{maxTrans} = \text{trans} \) then

17:     \( \text{total} \leftarrow \text{total} + \theta[\text{trans}] \cdot S[\text{node}] \)

18:     \( \text{nodeCount} \leftarrow \text{nodeCount} + 1 \)

19: end if

20: end for

21: \( i \leftarrow i + 1 \)

1: procedure \( \text{STIR MATRIX} \)

   INPUT \( \theta, \epsilon \)

   OUTPUT stirred \( \theta \)

2: for \( i \leftarrow 1 \) to \( |\theta| \) do

3: \( \theta[i] \leftarrow \theta[i] + \text{rand}(-\epsilon, \epsilon) \)

made of one node in the graph only; thus, filtration i this module decides whether to exclude a whole oneTrans locus (and node) or not. That takes place using a sampling-inspired model that classifies oneTrans loci into two categories: reliable (FP) which should be included, and unreliable (FZ) which should be filtered. The reliability of each locus is determined by the number of BLAST hits its only transcript receives with respect to a reference-based assembly considered as the GOLD standard. The details of the GOLD standard and all samples that were used appear in the next section.

When the pipeline needs to filter oneTrans loci of a new sample, the organisms in this sample are frequently unknown. The trained model thus decides based on the coverage of this transcript compared to the data used to train the model. The training data of the model are obtained from samples whose dominant organisms are known and tested. Trivially, the larger the training data, the better. A training dataset is made up of two different distributions: one for loci whose transcripts had zero BLAST hits with respect to the known GOLD standard, and another for ones with 1 or more hits. Each

---

**Fig. 2. Phases of multiTrans module of the pipeline**

In the E step (lines 8 to 10) of Algorithm 1, the likelihood of each node to belong to each transcript is calculated as per the logistic sigmoid function \( \delta \). A node is more likely to belong to transcripts with closer collective scores to its individual score. That is expressed as a higher probability by logistic sigmoid function. In the M step (lines 11 to 20), each transcript looks for to belong to each transcript is calculated as per the logistic sigmoid function. In the M step (lines 11 to 20), each transcript looks for its individual score and the final collective scores in to belong to predicted transcript and all samples that were used appear in the next section. The trained model thus decides whether to exclude a whole oneTrans locus (and node) or not. That takes place using a sampling-inspired model that classifies oneTrans loci into two categories: reliable (FP) which should be included, and unreliable (FZ) which should be filtered. The reliability of each locus is determined by the number of BLAST hits its only transcript receives with respect to a reference-based assembly considered as the GOLD standard. The details of the GOLD standard and all samples that were used appear in the next section.

When the pipeline needs to filter oneTrans loci of a new sample, the organisms in this sample are frequently unknown. The trained model thus decides based on the coverage of this transcript compared to the data used to train the model. The training data of the model are obtained from samples whose dominant organisms are known and tested. Trivially, the larger the training data, the better. A training dataset is made up of two different distributions: one for loci whose transcripts had zero BLAST hits with respect to the known GOLD standard, and another for ones with 1 or more hits. Each

---

**Algorithm 1 EM algorithm**

1: procedure EM

   INPUT number of filtered transcripts \( F \),
   node scores \( S \), set of nodes \( N \)

   OUTPUT Collective transcript scores \( \theta \)

2: initialize \( \theta[F] \)

3: initialize \( \theta[[N][|F|]] \) % likelihood matrix %

4: \( \text{MAXITER} \leftarrow 25 \)

5: \( i \leftarrow 1 \)

6: while \( i \leq \text{MAXITER} \) do

7: \( \text{STIR MATRIX}() \)

8: for node in \( N \) do

9:   for \( \text{trans} \leftarrow 1 \) to \( F \) do

10:      \( \theta[\text{trans}] \leftarrow \delta(S[\text{node}] - \theta[\text{trans}]) \)

11: end for

12: end for

13: \( \text{nodeCount} \leftarrow 0 \)

14: for node in \( N \) do

15:   \( \text{maxTrans} \leftarrow \max(\theta[\text{node}][\text{trans}]) \)

16:   if \( \text{maxTrans} = \text{trans} \) then

17:     \( \text{total} \leftarrow \text{total} + \theta[\text{trans}] \cdot S[\text{node}] \)

18:     \( \text{nodeCount} \leftarrow \text{nodeCount} + 1 \)

19: end if

20: end for

21: \( i \leftarrow i + 1 \)

1: procedure \( \text{STIR MATRIX} \)

   INPUT \( \theta, \epsilon \)

   OUTPUT stirred \( \theta \)

2: for \( i \leftarrow 1 \) to \( |\theta| \) do

3: \( \theta[i] \leftarrow \theta[i] + \text{rand}(-\epsilon, \epsilon) \)
of these distributions is created with respect to the average coverage values of transcript nodes in these loci. Then, both distributions are divided into well-defined buckets. The pipeline, based on the training data (the distributions), decides to select a transcript that lies in bucket $i$, $T$, with a probability of $p(T_i)$ defined in equation (2), where $FZ_i$ is the frequency of transcripts in the same bucket $i$ that got no hits in the training data and $FP_i$ is the frequency of transcripts that had one or more hits in the same bucket.

$$p(T_i) = \frac{2FZ_i}{FZ_i + FP_i} \quad (2)$$

For example, let the coverage of oneLoci transcripts ranges between 1 and 50 and the distribution is split into 10 buckets: [0, 5], [5,10], ..., [45,50]. A new transcript with coverage 4 lies in the first bucket ([0,5]). The pipeline checks how many reliable transcripts in the training data lie in the first bucket and how many unreliable ones (with no BLAST hits) do. Then, it chooses to filter or keep the transcripts according to Eq. 2. Clearly, the larger the ratio between number of reliable and unreliable transcripts in a certain bucket, the easier it is for the pipeline to decide whether to keep or disregard the new transcript. In practice, some buckets had close FP and FZ frequencies and thus were harder for the model. Other ones were easier with a considerable difference. One of the 6 distributions used in the training data is shown in Fig. 3.

![Fig. 3. Sample Average Coverage distribution used to train the pipeline’s second module](image)

In bucket 1, the ratio $\frac{FZ}{FP}$ in bucket 1 is $\approx \frac{1}{2}$ while it is $\approx \frac{2}{3}$ in bucket 3. Due to this large difference in ratio, it should be easier for the pipeline to predict whether to include or filter a transcript with a coverage value that lies in bucket 3 than in 1. Easiness in this context is expected to increase the accuracy of the pipeline when classifying oneLoci loci. An easier classification is one with less risk of excluding a transcript that should be included or vice versa. In all distributions obtained from all 6 samples used to collect training data, FP (frequency of reliable transcripts) was always considerably higher than FZ (frequency of unreliable transcripts) in all of the buckets.

The goal of this part of the pipeline is to filter the unreliable transcripts. Thus, for the pipeline to be effective, the filtered set of oneLoci loci should have higher accuracy than the original whole (unfiltered) one. Ideally, the pipeline should not exclude a reliable transcript. However, in practice, some of these transcripts will be filtered but their number should be considerably less than unreliable filtered ones. The higher the difference between these two filtered types (to the favor of unreliable filtered ones), the more effective this pipeline module is.

To run the pipeline, only one python script pipeline.py is needed beside three files produced by Oases: LastGraph that contains details about the final De Bruijn graph, stats.txt that includes statistics about all produced transcripts (especially their lengths and coverage values), and contig-ordering.txt which is the most important file that contains all produced transcripts clustered with respect to loci and graph nodes.

If the user needs to evaluate the accuracy of the filtered assembly while knowing the dominant species of in the metatranscriptomic sample under study, the python script blastAll.py should be used along with the reference genomes (if known) and the pipeline in [25]. That will allow running BLAST with filtered transcripts as queries and reference-based transcripts as target database. Then, pipeline.py contains procedures that reports accuracy and precision of the filtered assembly.

A third general purpose script called sequenceMetrics.py is available with the pipeline. It takes a FASTA file with one or more transcripts as an input and reports the following statistics on the transcripts collection in the file: minimum, maximum, median and average sequence lengths in addition to the N50 and L50 values. L50 is the number of sequences in the files whose length is greater than or equal to that of N50.

## 5 RESULTS

In each experiment, a metatranscriptomic sample is assembled using both reference-based and Oases (de novo). Samples are obtained from [16] and are all dominated by *Ecoli K12*, *Pmarinus*, *Rsphearo Boides chr1*, and *Rsphearo Boides chr2*. Knowing these species helped to build the GOLD standard made up of the reference-based assemblies. These GOLD transcripts constitute the BLAST database. The accuracy of the original assemblies by Oases and the ones filtered by the pipeline are measured according to the number of transcripts that get one or more BLAST hits with an E-value less than or equal to $e^{-30}$. As the pipeline filters Oases original assemblies, the number of transcripts in the filtered assemblies is bounded and often considerably smaller than that in the original one. If the pipeline was able to increase the accuracy while decreasing the number of transcripts, that would be a big advantage as assemblies become smaller in size (thus more focused and controllable) yet higher in quality. That proved to be the case in most experiments.

GOLD standard transcripts are obtained after running bwa-0.7.9a [20] and samtools0.1.19 [10] using the four reference genomes of dominant species in the samples. A blastn-2.2.29+ [14] database made up of these transcripts is created. Oases transcripts are then run as BLAST queries and results are reported accordingly. Command-based BLAST was run on Linux machines in all experiments. For Oases transcripts, 3 different kmer sizes (25, 27 and 29) were chosen before merging transcripts with respect to a 27-base kmer. Most multiTrans loci transcripts obtained 1 to 3 BLAST hits. Few outliers got up to 25 hits.

Original Oases assemblies ranged in size from 20443 to 21245 transcripts. In all of these assemblies, transcripts from oneTrans loci formed about 55% of the total while multiTrans ones formed the rest (45%). Filtration by the pipeline helped reduce the size by 5000+ transcripts. The user can partially control the number of filtered transcripts. That is mainly because $F$, which is the number of transcripts to remain in each multiTrans locus, is given as an input of the EM algorithm in the multiTrans part of the pipeline and can be pre-defined by the user for each locus.

In 5 out of the 6 experiments, the pipeline proved to be effective and able to increase accuracy and decrease the size of the assembly. Accuracy improvements range between 0.03% (sample S406) and 5.92% (sample S380). Results are shown in Fig. 4. Thousands of transcripts are filtered from each of the samples’ assemblies. Furthermore, the total precision of pipeline-filtered assemblies are always higher than accuracy values and range between 78.70% (S406) and 85.25% (S380). All values are shown in Table 1. In most experiments, the accuracy and precision values of the pipeline are very close. However, precision was around 10% or higher than accuracy for some samples (such as S403). In addition, it is worth noting
that the pipeline performed better at filtering multiTrans loci than oneTrans with respect to both precision and accuracy.

Table 1. Precision values of pipeline-filtered assemblies

<table>
<thead>
<tr>
<th>Sample</th>
<th>Precision (%)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S380</td>
<td>85.25</td>
<td>+0.01</td>
</tr>
<tr>
<td>S392</td>
<td>79.13</td>
<td>+0.2</td>
</tr>
<tr>
<td>S398</td>
<td>80.82</td>
<td>+0.07</td>
</tr>
<tr>
<td>S402</td>
<td>80.15</td>
<td>+0.09</td>
</tr>
<tr>
<td>S403</td>
<td>82.43</td>
<td>+9.56</td>
</tr>
<tr>
<td>S406</td>
<td>78.07</td>
<td>+3.71</td>
</tr>
</tbody>
</table>

while reducing the size of the original assembly by 5000+ transcripts and increasing the accuracy with 5.9+. Furthermore, in the only experiment where the accuracy went down, the difference was very small (-0.91%) compared to the size reduction that the pipeline led to for that assembly (5000+ less transcripts). Having smaller yet more or closely accurate filtered de novo assemblies is expected to utilize all advantages of de novo methods and minimize or even nullify those of reference-based ones. One main advantage of that is the ability to obtain high quality assemblies either without knowing the dominant species in metatranscriptomic samples or without having the reference genomes in such samples available. This pipeline is a step that proves to be helpful in that context.

Despite the fact that accuracy was the main quality metric chosen for all experiments, it is worth noting that the execution time for running the pipeline on a new sample is reasonable. The original Oases assembly was made up of 13000+ loci and 25000+ transcripts in all experiments and its running time was in less than an hour. The execution time of the pipeline is not far from that value. On average, running both modules of the pipeline to filter all oneTrans and multiTrans loci takes less than 2.5 hours per sample. The main reason behind using this pipeline by a new user is to improve the original assembly created by Oases. However, if the GOLD standard is available (transcripts obtained using bwa and sam tools against the available reference genomes), the accuracy (and precision) of the filtered pipeline can be tested in 1 to 2 hours for large metatranscriptomic samples such as the ones used in the experiments. That includes building the BLAST database, running BLAST against it using each Oases transcript and then reporting results.

The precision of the pipeline proved to always be - and sometimes considerably - higher than accuracy. Having high precision helps deploy the pipeline-filtered assemblies in sensitive applications that require most reliable transcripts to not be excluded. Having smaller assemblies that include most of these transcripts helps such applications reduce their execution time while maintaining comparable quality of their results.

A key factor of the pipeline is its usability. The only input needed is a FASTQ metatranscriptomic sample. The user then has to choose the input parameter \( F \) for each locus size or can use the default values. That makes the pipeline ready to run by executing one command only. All files that contain reported statistics, distributions, and details about filtered assembly are automatically created by the pipeline. The pipeline’s workflow was robustly tested for a high number of runs to guarantee its high level and straightforward usability. In addition, the script blastAll.py allows the user to further
study Oases assemblies by choosing between different kinds of statistics that may be applied to any Oases assembly. That can be done by changing the value of an argument in the command as explained in the manual. These statistics include the distribution of multiplicity value of the De Bruijn graph nodes, the number of nodes per transcript distribution, the number of transcripts per locus distribution, and other details that may be of the user’s concern if the GOLF standard assembly is available. A self-described file that contains tested R commands to plot these distributions is available as well.

The pipeline perform better when filtering multiTrans loci than oneTrans one. Two possible reasons could be behind that issue: the randomized sampling used in oneTrans filtering and the sophisticated multi-step filtration of multiTrans ones. Indeed, the first part was later integrated with the pipeline. Some suggestions for further improving its performance are mentioned in the next section.

7 FUTURE WORK

The fact that the pipeline is divided into two separate modules makes it easily parallelizable. A parallel implementation of the pipeline is easily possible due to the separation between oneTrans and multi-Trans loci. That’s expected to further speedup the filtration of Oases assemblies and makes the pipeline a potential postprocessing step after each Oases run. In addition, a number of steps that may further improve oneTrans loci transcripts filtration can be added. Two of these are (1) combining coverage and length average values of such transcripts to perform the filtering accordingly and (2) accumulating more training data. Despite the fact that the logistic sigmoid function proved to perform best against many other models in the EM algorithm of Algorithm 1, other models can be tested in an attempt to improve the performance of the algorithm. Adding a simple GUI that would also help the user run the scripts with the click of a button instead of writing down the command. That is especially helpful for Biology specialists who don’t have any experience with command-based execution of software systems.

Graph manipulation steps can be added to the pipeline to resolve transcripts predicted by the pipeline after running the EM algorithm. That way, the pipeline can produce its own set of transcripts without any need of waiting for Oases to generate transcripts. The graph and statistics files will be the only ones taken as an input. The quality of such transcripts will depend on the graph manipulation steps involved and should be compared to Oases original assemblies. Furthermore, the pipeline is not restricted to filter Oases transcripts and can be further extended to filter assemblies produced by other De Bruijn graph assemblers such as Velvet, Trinity, and ABySS. Changing the score function in equation (1) allows the pipeline to work on assemblies generated by MIRA as well.

8 CONCLUSION

In this paper, we presented a pipeline that filters metatranscriptomic assemblies which are based on De Bruijn graphs. The pipeline is made up of two integral modules that process two main types of transcripts generated by Oases, one of the most well-known RNA-seq de novo assemblers. In the first module, an EM algorithm predicts new transcripts and filters part of the original de novo assembly accordingly. In the second, a sample approach based on training data accumulated runs is adopted. Experiments showed that the pipeline helps in increasing the accuracy of the assemblies while considerably reducing their size.

REFERENCES