SOFTWARE TOOL ARTICLE

ViennaNGS: A toolbox for building efficient next-generation sequencing analysis pipelines [version 1; peer review: 3 approved with reservations]

Michael T. Wolfinger¹-³, Jörg Fallmann¹, Florian Eggenhofer¹, Fabian Amman¹,²,³,⁴

¹Institute for Theoretical Chemistry, University of Vienna, Währingerstraße 17, A-1090, Vienna, Austria
²Center for Integrative Bioinformatics Vienna, Max F. Perutz Laboratories, University of Vienna, Medical University of Vienna, Dr. Bohr-Gasse 9, A-1030 Vienna, Austria
³Department of Biochemistry and Molecular Cell Biology, Max F. Perutz Laboratories, University of Vienna, Dr. Bohr-Gasse 9, 1030 Vienna, Austria
⁴Department of Chromosome Biology, Max F. Perutz Laboratories, University of Vienna, Medical University of Vienna, Dr. Bohr-Gasse 9, A-1030 Vienna, Austria

Abstract
Recent achievements in next-generation sequencing (NGS) technologies lead to a high demand for reusable software components to easily compile customized analysis workflows for big genomics data. We present ViennaNGS, an integrated collection of Perl modules focused on building efficient pipelines for NGS data processing. It comes with functionality for extracting and converting features from common NGS file formats, computation and evaluation of read mapping statistics, as well as normalization of RNA abundance. Moreover, ViennaNGS provides software components for identification and characterization of splice junctions from RNA-seq data, parsing and condensing sequence motif data, automated construction of Assembly and Track Hubs for the UCSC genome browser, as well as wrapper routines for a set of commonly used NGS command line tools.

Keywords
Perl, next generation sequencing, RNA-seq, read mapping, pipelines
Introduction

Next-generation sequencing (NGS) technologies have influenced both our understanding of genomic landscapes as well as our attitude towards handling big biological data. Emerging functional genomics methods based on high-throughput sequencing allow investigation of highly specialized and complex scientific questions, which continuously poses challenges in the design of analysis strategies. Moreover, the demand for efficient data analysis methods has dramatically increased. While a typical NGS analysis workflow is built on a cascade of routine tasks, individual steps are often specific for a certain assay, e.g. depend on a particular sequencing protocol.

A set of NGS analysis pipelines are available for general\textsuperscript{12}, and specialized assays such as de-novo motif discovery\textsuperscript{1}. While these tools mostly cover the elementary steps of an analysis workflow, they often represent custom-tailored solutions that lack flexibility. Web-based approaches like Galaxy\textsuperscript{6} cover a wide portfolio of available applications, however they do not offer enough room for power users who are used to the benefits of the command line.

The recently published HTSeq framework\textsuperscript{1} as well as the biotoolbox package provide library modules for processing high-throughput data. While both packages implement NGS analysis functionality in a coherent manner, we encountered use cases that were not covered by these tools.

Motivation

The motivation for this contribution emerged in the course of the research consortium “RNA regulation of the transcriptome” (Austrian Science Fund project F43), which brings together more than a dozen experimental groups with various thematic backgrounds. In the line of this project it turned out that complex tasks in NGS analysis could easily be automated, whereas linking individual steps was very labour-intensive. As such, it became apparent that there is a strong need for modular and reusable software components that can efficiently be assembled into different full-fledged NGS analysis pipelines.

We present ViennaNGS, a Perl distribution that integrates high-level routines and wrapper functions for common NGS processing tasks. ViennaNGS is not an established pipeline per se, it rather provides tools and functionality for the development of NGS pipelines. It comes with a set of utility scripts that serve as reference implementation for most library functions and can readily be applied for specific tasks or integrated as-is into custom pipelines. Moreover, we provide extensive documentation, including a dedicated tutorial that showcases core features of the software and discusses common application scenarios.

Development of the ViennaNGS suite was triggered by two driving forces. On the one hand we wanted to return to the open source community our own contribution, which itself is heavily based and dependent on open source software. On the other hand, beside “open science” we advocate for the concept of “reproducible science”\textsuperscript{10}. Unfortunately, and to some extent surprising, bioinformatics analyses are often not fully reproducible due to inaccessibility of tools (keyword “in-house script”) or software versions used. It is therefore essential to ensure the entire chain of reproducibility from data generation to interpretation in the analysis of biological data.

Methods

The major design consideration for the ViennaNGS toolbox was to make available modular and reusable code for NGS processing in a popular scripting language. We therefore implemented thematically related functionality in different Perl modules under the Bio namespace (Figure 1), partly building on BioPerl\textsuperscript{7} and the Moose object framework. Our focus is on consistent versioning, facilitated through Github hosting. In addition, ViennaNGS releases are available via the Comprehensive Perl Architecture Network (CPAN), thereby enabling users to get back to previous versions at any time in order to reenact conclusions drawn from shared biological data.

ViennaNGS has been designed to close gaps in established analysis workflows by covering a wide range of processing steps from raw data to data visualization. In the following we introduce individual ViennaNGS components and describe their main functionality.

BAM manipulation and filtering

Once mapped to a reference genome, NGS data is typically stored in the widely used SAM/BAM file format. BAM is a binary format, which can easily be converted into text-based SAM format via samtools\textsuperscript{8} for downstream analysis. However, modern NGS assays produce hundreds of millions of reads per sample, hence SAM files tend to become excessively large and can have a size of several hundred gigabytes. Given that storage resources are always limited, strategies to efficiently retrieve mapping information from BAM format are an asset. To accomodate that, we provide functionality for querying global mapping statistics and extracting specific alignment information from BAM files directly.

Figure 1. Schematic overview of ViennaNGS components. Core modules can be combined in a flexible manner to address individual analysis objectives and experimental setup.
ViennaNGS::BamStat extracts both qualitative and quantitative information from BAM files, i.e., the amount of total alignments, aligned reads, as well as uniquely and multi mapped reads. Numbers are reported individually for single-end reads, paired-end fragments and pairs missing a mate. Quality-wise ViennaNGS::BamStat collects data on edit distance in the alignments, fraction of clipped bases, fraction of matched bases, and quality scores for entire alignments. Subsequently, ViennaNGS::BamStatSummary compares different samples in BAM format and illustrates results graphically. Summary information is made available in CSV format to facilitate downstream processing.

Efficient filtering of BAM files is among the most common tasks in NGS analysis pipelines. Building on the BioSamTools distribution, ViennaNGS::Bam provides a set of convenience routines for rapid manipulation of BAM files, including filters for unique and multiple alignments as well as functionality for splitting BAM files by strand, thereby creating two strand-specific BAM files. Results can optionally be converted to BedGraph or BigWig formats for visualization purposes.

**Genomic annotation**

Proper handling of genomic intervals is essential for NGS analysis pipelines. Several feature annotation formats have gained acceptance in the scientific community, including BED, GTF, GFF, etc., each having its particular benefits and drawbacks. While annotation for a certain organism is often only available in a specific format, interconversion among these formats can be regarded a routine task, and a pipeline should be capable of processing as many formats as possible.

We address this issue at different levels. On the one hand, we provide ViennaNGS::AnnoC, a lightweight annotation converter for non-spliced genomic intervals, which can be regarded a simple yet powerful solution for conversion of bacterial annotation data. On the other hand we have developed an abstract representation of genomic features via generic Moose-based classes, which provide functionality for efficient manipulation of BED4, BED6, BED12 and GTF/GFF elements, respectively, and allow for BED format conversion facilitated by ViennaNGS::MinimalFeature represents an elementary genomic interval, characterized by chromosome, start, end and strand. ViennaNGS::Feature extends ViennaNGS::MinimalFeature by two attributes, name and score, thereby creating a representation of a single BED6 element. ViennaNGS::FeatureLine pools a set of ViennaNGS::Feature objects via an array reference. All intervals of interest can be covered by a ViennaNGS::FeatureLine object, which holds a hash of references to ViennaNGS::FeatureChain objects (Figure 2).

This framework can handle annotation data by providing abstract data representations of genomic intervals such as exons, introns, splice junctions etc. It allows for efficient description and manipulation of genomic features up to the level of transcripts (Figure 3). Conversely, it is highly generic and can be extended to hierarchically higher levels such as genes composed of different transcript isoforms or clusters of paralogous genes.

![Figure 2. Class diagram illustrating the relations among generic Moose classes which are used as abstract representations of genomic intervals (only attributes are shown).](image)

![Figure 3. Schematic representation of genomic intervals classes in terms of their corresponding feature annotation. Simple intervals ("features") are characterized by Bio::ViennaNGS::Feature objects (bottom box). At the next level, Bio::ViennaNGS::FeatureChain bundles these, thereby maintaining individual annotation chains for e.g. UTRs, exons, introns, splice junctions, etc. (middle box). The topmost level is given by Bio::ViennaNGS::FeatureLine objects, representing individual transcripts.)](image)
Visualization

Another cornerstone of NGS analysis pipelines is graphical representation of mapped sequencing data. In this context a standard application is visualization of Chip-seq peaks or RNA-seq coverage profiles. The latter are typically encoded in Wiggle format, or its indexed binary variant, BigWig, which can readily be displayed within a genome browser. In the same line, genomic annotation and intervals are often made available in BigBed format, an indexed binary version of BED. ViennaNGS::Util comes with wrapper routines for automated conversion from common formats like BAM to BigWig or BED to BigBed via third-party utilities. In addition, we have implemented interfaces for a selection of BEDtools components as well as a collection of auxiliary routines.

The UCSC genome browser allows to display potentially large genomic data sets, that are hosted at Web-accessible locations by means of Track Hubs. On a more general basis this even works for custom organisms that are not supported by default through the UCSC genome browser, via Assembly Hubs. A typical use case is visualization of genomic annotation, RNA-seq coverage profiles and Chip-seq peaks for Arabidopsis thaliana (which is not available through the generic UCSC browser) via a locally hosted Assembly Hub. ViennaNGS::UCSC provides all relevant routines for automatic construction of Assembly and Track Hubs from genomic sequence and/or annotation. Besides automated Assembly and Track Hub generation, we support deployment of custom organism databases in local mirrors of the UCSC genome browser.

Gene expression and normalization

RNA-seq has become a standard approach for gene and transcript quantification by means of measuring the relative amount of RNA present in a certain sample or under a specific condition, thus ideally providing a good estimate for the relative molar concentration of RNA species. Simple “count-based” quantification models assume that the total number of reads mapping to a region can be used as a proxy for RNA abundance. A good measure for transcript abundance is ideally as closely proportional to the relative molar concentration of a RNA species as possible. Various measures have been proposed, one of the most prominent being RPKM (reads per kilobase per million). It accounts for different transcript lengths and sequencing depth by normalizing by the number of reads in a specific sample, divided by $10^6$. It has, however, been shown that RPKM is not appropriate for measuring the relative molar concentration of a RNA species due to normalization by the total number of reads.

Alternative measures that overcome this shortcoming have been suggested, e.g. TPM (transcript per million) (Equation 1). Here, rather than normalizing by the total number of mapped reads, a proxy for the total number of transcript samples considering the sequencing reads per gene $r_g$ is used for normalization (Equation 2). The variable $l_f$ is the read length and $l_f$, the feature length of a gene region $g$. Consequently, $T$ can be computed by summing over the set of all genes $G$.

$$TPM_g = \frac{r_g \times r_l}{l_f \times T \times 10^6}$$

$$T = \sum_{g \in G} r_g \times r_f$$

We provide routines for the computation of TPM values for genomic intervals from raw read counts within ViennaNGS::Expression.

Characterization of splice junctions

ViennaNGS::SpliceJunc addresses a more specific problem, namely characterization of splice junctions which is becoming increasingly relevant for understanding alternative splicing. This module provides code for identification and characterization of splice junctions from short read mappers. It can detect novel splice junctions in RNA-seq data and generate visualization files. While we have focused on processing the output of segemehl, the module can easily be extended for other splice-aware split read mappers.

Documentation and tutorial

The ViennaNGS suite comes with extensive documentation based on Perl’s POD system, thereby providing a single documentation base which is available through different channels, e.g. on the command line via the perldoc utility or on the Web via CPAN. Moreover, we provide ViennaNGS::Tutorial to guide prospective users through the development of basic NGS analysis pipelines. The tutorial is split into different chapters, each covering a common use case in NGS analysis and describing a possible solution.

Utilities

The ViennaNGS suite comes with a collection of complementary executable Perl scripts for accomplishing routine tasks often required in NGS data processing. These command line utilities serve as reference implementations of the routines implemented in the library and can readily be used for atomic tasks in NGS data processing. Table 1 lists the utilities and gives a short description of their functionality.

Discussion

ViennaNGS is a comprehensive software library for rapid development of custom NGS analysis pipelines. We have successfully applied its components in the course of an ongoing, large scale collaboration project focusing on RNA regulation. It has been used with different genomics assays in a wide range of biological systems, including human, plants and bacteria. While we have primarily applied ViennaNGS in combination with the short read aligner segemehl, it has also been used with Tophat output
very recently in a large scale transcriptome study of Ebola and Marburg virus infection in human and bat cells (Hölzer et al., unpublished data). Moreover, ViennaNGS will be used for automated UCSC genome browser integration in an upcoming version of TSSAR\(^1\), a recently published approach for characterization of transcription start sites from dRNA-seq data.

ViennaNGS is actively developed and its functionality is constantly extended. In this line, we encourage the scientific community to contribute patches and novel features.

**Data availability**

Input data for the ViennaNGS tutorial is available from http://rna.tbi.univie.ac.at/ViennaNGS

**Software availability**

The ViennaNGS distribution is available through the Comprehensive Perl Architecture Network (CPAN) at and GitHub.

**Software access**

http://search.cpan.org/dist/Bio-ViennaNGS

**Latest source code**

https://github.com/mtw/Bio-ViennaNGS

---

**Table 1. Overview of the complementary utilities shipped with ViennaNGS.** While some of these scripts are re-implementations of functionality available elsewhere, they have been developed primarily as reference implementation of the library routines to help prospective ViennaNGS users getting started quickly with the development of custom pipelines.

<table>
<thead>
<tr>
<th>Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assembly_hub_constructor.pl</td>
<td>Construct Assembly Hubs for UCSC genome browser visualization</td>
</tr>
<tr>
<td>bam_quality_stat.pl</td>
<td>Compute mapping/quality statistics along with publication-ready figures</td>
</tr>
<tr>
<td>bam_split.pl</td>
<td>Split BAM files by strand</td>
</tr>
<tr>
<td>bam_to_bigwig.pl</td>
<td>Produce BigWig coverage profiles from BAM files for visualization</td>
</tr>
<tr>
<td>bam_uniq.pl</td>
<td>Filter uniquely and multi mapped reads from BAM files</td>
</tr>
<tr>
<td>bed2bedGraph.pl</td>
<td>Convert BED to (strand specific) bedGraph format</td>
</tr>
<tr>
<td>extend_bed.pl</td>
<td>Extend genomic intervals in BED format at the 5’, 3’, or both ends</td>
</tr>
<tr>
<td>gff2bed.pl</td>
<td>Convert bacterial RefSeq GFF3 annotation to BED12 format</td>
</tr>
<tr>
<td>kmer_analysis.pl</td>
<td>Count k-mers of predefined length in FastQ and Fasta files</td>
</tr>
<tr>
<td>MEME_xml_motif_extractor.pl</td>
<td>Compute basic statistics from MEME XML output</td>
</tr>
<tr>
<td>newUCSCdb.pl</td>
<td>Create a new genome database in a local UCSC genome browser instance</td>
</tr>
<tr>
<td>normalize_multicov.pl</td>
<td>Compute normalized expression data in TPM from read counts</td>
</tr>
<tr>
<td>sj_visualizer.pl</td>
<td>Convert splice junctions in segemehl BED6 splice junction format to BED12</td>
</tr>
<tr>
<td>splice_site_summary.pl</td>
<td>Identify and characterize splice junctions from RNA-seq data</td>
</tr>
<tr>
<td>track_hub_constructor.pl</td>
<td>Construct Track Hubs for UCSC genome browser visualization</td>
</tr>
<tr>
<td>trim_fastq.pl</td>
<td>Trim sequence and quality fields in FastQ format</td>
</tr>
</tbody>
</table>

---

**Archived source code as at the time of publication**

http://dx.doi.org/10.5281/zenodo.15088

**License**

The Perl 5 License

---

**Author contributions**

MTW, JF, FE, FA designed and implemented the software. MTW and FA wrote the manuscript. All authors approved the final manuscript.

**Competing interests**

No competing interests were disclosed.

**Grant information**

This work was funded by the Austrian Science Fund (FWF projects F43 to MTW, FA and FE) and the Research Platform “Decoding mRNA decay in inflammation” by the University of Vienna to JF.

**Acknowledgments**

A preprint version of this article can be found on BioRxiv: http://dx.doi.org/10.1101/013011
References

   PubMed Abstract | Publisher Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
   Reference Source
   PubMed Abstract | Publisher Full Text | Free Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
   PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
   Reference Source
    PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
    PubMed Abstract | Publisher Full Text | Free Full Text
Open Peer Review

Current Peer Review Status:  

Version 1

Reviewer Report 24 April 2015

https://doi.org/10.5256/f1000research.6600.r8396

© 2015 Voß B. This is an open access peer review report distributed under the terms of the Creative Commons Attribution Licence, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Björn Voß  
University of Freiburg, Stuttgart, Germany

In their manuscript about ViennaNGS the authors describe a set of perl modules and scripts that is useful to build pipelines for NGS data analysis. A key motivation for this is to promote reproducible science, especially with respect to medium-level users, who often create "in-house scripts" for data analysis, which are rarely publicly available. This target community distinguishes ViennaNGS from related approaches, such as Galaxy. The contribution is, thus, relevant and has the potential to serve as a basis for future developments in NGS analysis pipelines. I tested the tutorials and some of the utility scripts and they worked fine. Nevertheless, I think the authors need to clarify some issues and can improve the presentation of their work.

Major Comments:

- The authors should point out clearly, what distinguishes ViennaNGS from other suites. In the end, they need to convince people to use ViennaNGS. For that it would be helpful to clearly state what is hard or even impossible to implement in one of the other systems (galaxy, HTSeq, ...) at best with real world examples.

- As stated in the title the aim of ViennaNGS is to ease the process of building NGS analysis pipelines. Unfortunately, exactly this aspect is more or less not mentioned in the main text. It would be interesting to know, especially for the data analysts with scripting experience, how such a pipeline looks like and why it is easier to build using ViennaNGS.

- I do not quite understand the explicit discussion of TPM and RPKM. The differences are extensively discussed in Wagner et al. (2012), which the authors can refer to.

- Similarly, the description of the accompanying utilities in Table 1 is of minor interest. I would suggest to mention them when the corresponding functionality is described in the main text, e.g., assembly_hub_constructor.pl in the paragraph on Visualization. Furthermore, the authors can explain one tool in detail to show how ViennaNGS pipelines are implemented.
BioPerl already provides modules to handle Annotation Features (Bio::SeqFeature), which at first glance seem to provide the same functionality as the ViennaNGS feature annotation classes. Why is there a need for an own class?

Minor Comments:

- An aspect that is becoming more and more important is parallelization. The authors should describe the possibilities of ViennaNGS to be used in cluster or massively parallel environments.

- The authors should make clear that for some/many tasks they use external tools, such as bedtools2, samtools and tools offered by the UCSC and that the user has to install them on its own. Of course, this is the same as for galaxy and others.

- I was wondering if ViennaNGS or its pipelines may be integrated into Galaxy. In this way the systems would complement and benefit from each other.

- At the end of the discussion the authors could provide actual functionalities that they are planning to integrate in the near future. This is interesting for potential users who are missing certain functionalities in the current release. An example is quality control of the raw sequencing data.

- P.5, Software availability: "at and" --> "and at"

**Competing Interests:** No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

---

**Reader Comment 06 Jul 2015**

**Michael T. Wolfinger**, University of Vienna, Austria

We would like to thank you for taking the time to review this manuscript, as well as for your helpful comments. We have addressed every issue raised here in a point-to-point manner and modified our manuscript accordingly at different places. We hope that the changes are satisfactory.

The authors should point out clearly, what distinguishes ViennaNGS from other suites. In the end, they need to convince people to use ViennaNGS. For that it would be helpful to clearly state what is hard or even impossible to implement in one of the other systems (galaxy, HTSeq,...) at best with real world examples.

We appreciate this comment and have added a paragraph in the Introduction, highlighting both the Moose-based object oriented design, and the Perl 6 compliance, which can be regarded as unique selling points of the ViennaNGS suite. ViennaNGS has been developed to provide a toolbox that helps users build their analysis pipelines in Perl, thus targeting researchers who are more literate in Perl than Python. However, ViennaNGS is an open platform and it should therefore be straightforward to implement ViennaNGS-based pipelines within e.g. Galaxy for experienced users.

As stated in the title the aim of ViennaNGS is to ease the process of building NGS analysis pipelines. Unfortunately, exactly this aspect is more or less not mentioned in the main text. It would be interesting to know, especially for the data analysts with scripting
experience, how such a pipeline looks like and why it is easier to build using ViennaNGS.

We have added a new section ‘Applications’ where the process of building custom pipelines is exemplified in terms of the ViennaNGS Tutorials and Utilities. The ViennaNGS Tutorials explain in detail how custom pipelines can be built for a set of real-world NGS applications.

I do not quite understand the explicit discussion of TPM and RPKM. The differences are extensively discussed in Wagner et al. (2012), which the authors can refer to.

The extensive discussion of TPM and RPKM, including all formulas, have been removed from the manuscript.

Similarly, the description of the accompanying utilities in Table 1 is of minor interest. I would suggest to mention them when the corresponding functionality is described in the main text, e.g., assembly_hub_constructor.pl in the paragraph on Visualization. Furthermore, the authors can explain one tool in detail to show how ViennaNGS pipelines are implemented.

We respectfully disagree and think that the ViennaNGS utilities should be mentioned in one place, given that they can be regarded, apart from the ViennaNGS Tutorials, as yet another set of example implementations of ViennaNGS library functions. Moreover, we have shifted the paragraph mentioning the Utilities into the Applications section.

BioPerl already provides modules to handle Annotation Features (Bio::SeqFeature), which at first glance seem to provide the same functionality as the ViennaNGS feature annotation classes. Why is there a need for an own class?

BioPerl and it’s associated modules are a fantastic toolbox for everyday bioinformatics work and we use them whenever applicable (e.g. via Bio::DB::Sam). In general the Bio::Seq and especially Bio::SeqFeature classes allow a multitude of operations on common biological features, their annotations and file formats. ViennaNGS was designed with strong focus on NGS analysis and easy portability to Perl 6. Given that the ViennaNGS feature annotation classes play a pivotal role in current and future development of the toolbox, we decided to implement Moose classes without introducing too many dependencies on existing BioPerl modules. We went for a design that specifically fits the needs of NGS analysis and stays as minimal as possible. In this sense, we do not see ViennaNGS in competition to BioPerl but as a boutique alternative for NGS data analysts.

An aspect that is becoming more and more important is parallelization. The authors should describe the possibilities of ViennaNGS to be used in cluster or massively parallel environments.

We appreciate this comment and have added a statement on parallelization of ViennaNGS-based pipelines into the Discussion. Our focus in the initial development phase of ViennaNGS has not been on parallelization, hence the code base has not been specifically designed for parallel processing in a cluster environment. It should, however, be straightforward to implement certain tasks in multithreaded pipelines.

The authors should make clear that for some/many tasks they use external tools, such as bedtools2, samtools and tools offered by the UCSC and that the user has to install them
on its own. Of course, this is the same as for galaxy and others.

A section listing all third party dependencies has been added to the main text.

I was wondering if ViennaNGS or its pipelines may be integrated into Galaxy. In this way the systems would complement and benefit from each other.

As mentioned earlier, since ViennaNGS is implemented purely in Perl, it should be straightforward for experienced users to integrate its functionalities into Galaxy, e.g., via the the Galaxy Tool Factory.

At the end of the discussion the authors could provide actual functionalities that they are planning to integrate in the near future. This is interesting for potential users who are missing certain functionalities in the current release. One example is quality control of the raw sequencing data.

The Discussion has been updated accordingly.

P.5, Software availability: "at and" --> "and at"

Done.

Competing Interests: No competing interests were disclosed.
whole human genome sequence would be helpful. For filtering comparisons, I suggest comparing with samtools or sambamba (https://github.com/lomereiter/sambamba). For quality control, comparisons to QualiMap (http://qualimap.bioinfo.cipf.es/) or bamtools (https://github.com/pezmaster31/bamtools) would be helpful.

• Similarly, it would be great to have benchmarking on annotation and BED manipulation tools in ViennaNGS. How does the functionality and timing compare with bedtools? You require and use bedtools for visualization, and it would be useful to clarify benefits and tradeoffs to using ViennaNGS versus interfacing directly with bedtools.

• How do you handle testing and validation of ViennaNGS tools and pipelines? I saw new tests for UCSC integration coming in during review, which is great. It would be nice to understand the process by which you ensure new development improves (or at least doesn't degrade) the biological results.

**Competing Interests:** No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

---

**Reader Comment 06 Jul 2015**

**Michael T. Wolfinger**, University of Vienna, Austria

Thank you very much for taking the time to review our manuscript. We appreciate your comments and have addressed every issue raised here in a point-to-point manner and modified our manuscript accordingly at different places. We hope that the changes are satisfactory.

Please include additional information about what is unique about ViennaNGS in the introduction. Currently it reads generally and is more about pointing about flaws in other software without saying what ViennaNGS provides. The motivation provides much of this text but it seems out of order relative to the introductory material.

Thank you very much for this comment, which is highly appreciated. We have re-arranged the Introduction and Methods sections and provide additional information on ViennaNGS’ unique selling points, specifically its object oriented design based on the Moose framework and consequently Perl 6 compliance.

Please provide benchmarks on your BAM manipulation tools relative other common tools. I don't think this needs to be extensive, but providing a summary of how they perform on a 100Gb 30x whole human genome sequence would be helpful. For filtering comparisons, I suggest comparing with samtools or sambamba (https://github.com/lomereiter/sambamba). For quality control, comparisons to QualiMap (http://qualimap.bioinfo.cipf.es/) or bamtools (https://github.com/pezmaster31/bamtools) would be helpful.

ViennaNGS has been designed as a toolbox for building NGS pipelines and does not do any SAM/BAM manipulation itself. For the latter we rely on Bio::DB::Sam, which uses the samtools library internally. Comparison against the mentioned tools is difficult since, to our knowledge, Perl
library internally. Comparison against the mentioned tools is difficult since, to our knowledge, Perl bindings for the mentioned tools are not available.

We have benchmarked the ViennaNGS tutorials and provide statistics on time and memory consumption in Table 1. For consistency we have applied the benchmarks to files smaller than the suggested 100Gb 30x coverage, since they are part of our tutorial pipeline and can readily be downloaded from our Web server at http://rna.tbi.univie.ac.at/ViennaNGS.

Similarly, it would be great to have benchmarking on annotation and BED manipulation tools in ViennaNGS. How does the functionality and timing compare with bedtools? You require and use bedtools for visualization, and it would be useful to clarify benefits and tradeoffs to using ViennaNGS versus interfacing directly with bedtools.

Here the same arguments concerning benchmarking given above apply. Wherever possible we use bedtools for BED manipulation rather than interfacing directly with BED files. The major benefit of using ViennaNGS versus interfacing directly with bedtools is to have data stored consistently in Moose objects which can be referenced throughout the toolbox. As for timing, we do not expect any impact since all bedtools utilities are called via Perl system calls, thus conserving the original bedtools functionality.

How do you handle testing and validation of ViennaNGS tools and pipelines? I saw new tests for UCSC integration coming in during review, which is great. It would be nice to understand the process by which you ensure new development improves (or at least doesn't degrade) the biological results.

We have added a paragraph outlining the ViennaNGS testing strategy. While we have not yet implemented testing on a global scale, the ViennaNGS::SpliceJunc and ViennaNGS::UCSC modules are currently tested automatically and tests for feature annotation classes will be added in the near future.

**Competing Interests:** No competing interests were disclosed.
Still, I feel the description of the software is rather too general and could be improved.

Major Comments:

The article lacks any benchmarking or presentation of an example analysis, making it difficult to put the software's performance in perspective with any of the other numerous tools already available. Important for NGS data analysis are specifications for the usage of computational resources (RAM, number of CPUs, processing time, space requirements) and how those scale up with the size of the data set (=number and size of data sets) or type of NGS data (genomic, RNAseq, ChIPseq, Bisulfite-Seq) - all of which are not mentioned. Similarly, the authors do not make any statement on the possibility of parallelization or adaption to cluster infrastructures.

Minor comments:

Although, truly RPKM has been shown to be inappropriate for measuring the relative molar concentration of a RNA species due to normalization by the total number of reads, it is still widely used. Computing RPMK values as well (optionally) as TPM would allow for comparison with other pipelines.

**Competing Interests:** No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

---

Reader Comment 06 Jul 2015

**Michael T. Wolfinger**, University of Vienna, Austria

Thank you very much for taking the time to review our manuscript and for your helpful comments. We have addressed raised issues here in a point-to-point manner, adjusted the text accordingly, and added the requested functionality to the library. We hope that these changes are satisfactory.

The article lacks any benchmarking or presentation of an example analysis, making it difficult to put the software's performance in perspective with any of the other numerous tools already available. Important for NGS data analysis are specifications for the usage of computational resources (RAM, number of CPUs, processing time, space requirements) and how those scale up with the size of the data set (=number and size of data sets) or type of NGS data (genomic, RNAseq, ChIPseq, Bisulfite-Seq) - all of which are not mentioned. Similarly, the authors do not make any statement on the possibility of parallelization or adaption to cluster infrastructures.

We appreciate this comment and have addressed the concerns at different places throughout the manuscript. These include:

- Benchmarking and presentation of examples: We have added a section 'Applications' where the process of building custom pipelines is exemplified in terms of the ViennaNGS Tutorials and Utilities. We provide coherent benchmarking data of computer resources required to run the Tutorial pipelines in Table 1. The ViennaNGS Tutorial pipelines have been specifically designed as example implementations of custom ViennaNGS-based analysis workflows. Data-intensive tasks, e.g. BED or BAM filtering, are mainly performed by system calls to third-party tools (bedtools and samtools, respectively), that work on BED
by system calls to third-party tools (bedtools and samtools, respectively), that work on BED or BAM files, regardless whether these originate from RNA-seq or other NGS assays.

- We appreciate the comment on the possibility of parallelization, which has also been raised by another reviewer, and have added a statement regarding parallelization of ViennaNGS-based pipelines into the Discussion. While the code base not been specifically designed for execution in a parallel environment, specific tasks such as splitting of BAM files can be parallelized trivially within custom ViennaNGS pipelines, provided sufficient IO resources are available.

- For consistency reasons we stuck to the supplementary shipped data, as included in the Tutorials, for benchmarking. Where applicable, we modified the Tutorials and tested their performance with increasing number of input file (e.g., Tutorial 0) or with increasing size of input file (e.g., Tutorial 2). Both tests showed, as expected, a linear relationship between input, memory and time consumption, respectively.

Although, truly RPKM has been shown to be inappropriate for measuring the relative molar concentration of a RNA species due to normalization by the total number of reads, it is still widely used. Computing RPMK values as well (optionally) as TPM would allow for comparison with other pipelines.

Thank you very much for this comment. We have added the possibility to compute RPKM alongside TPM within the Bio::ViennaNGS::Expression module and updated the normalize_multicov.pl utility accordingly. Modified versions of the mentioned software are available in Bio::ViennaNGS v0.15.

**Competing Interests:** No competing interests were disclosed.