pyrpipe: a python package for RNA-Seq workflows

Urminder Singh  
*Iowa State University*, usingh@iastate.edu

Jing Li  
*Iowa State University*, jingli@iastate.edu

Arun Seetharam  
*Iowa State University*, arnstrm@iastate.edu

Eve Syrkin Wurtele  
*Iowa State University*, mash@iastate.edu

Follow this and additional works at: https://lib.dr.iastate.edu/gdcb_las_pubs

Part of the Bioinformatics Commons, Cell and Developmental Biology Commons, and the Computational Biology Commons

The complete bibliographic information for this item can be found at https://lib.dr.iastate.edu/gdcb_las_pubs/239. For information on how to cite this item, please visit http://lib.dr.iastate.edu/howtocite.html.

This Article is brought to you for free and open access by the Genetics, Development and Cell Biology at Iowa State University Digital Repository. It has been accepted for inclusion in Genetics, Development and Cell Biology Publications by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Abstract
Implementing RNA-Seq analysis pipelines is challenging as data gets bigger and more complex. With the availability of terabytes of RNA-Seq data and continuous development of analysis tools, there is a pressing requirement for frameworks that allow for fast and efficient development, modification, sharing and reuse of workflows. Scripting is often used, but it has many challenges and drawbacks. We have developed a python package, python RNA-Seq Pipeliner (pyrpipe) that enables straightforward development of flexible, reproducible and easy-to-debug computational pipelines purely in python, in an object-oriented manner. pyrpipe provides high level APIs to popular RNA-Seq tools. Pipelines can be customized by integrating new python code, third-party programs, or python libraries. Researchers can create checkpoints in the pipeline or integrate pyrpipe into a workflow management system, thus allowing execution on multiple computing environments. pyrpipe produces detailed analysis, and benchmark reports which can be shared or included in publications. pyrpipe is implemented in python and is compatible with python versions 3.6 and higher. All source code is available at https://github.com/urmi-21/pyrpipe; the package can be installed from the source or from PyPi (https://pypi.org/project/pyrpipe). Documentation is available on Read the Docs (http://pyrpipe.rtfd.io).

Disciplines
Bioinformatics | Cell and Developmental Biology | Computational Biology | Genetics and Genomics

Comments
This preprint is made available through bioRxiv at, doi: 10.1101/2020.03.04.925818.
Implementing RNA-Seq analysis pipelines is challenging as data gets bigger and more complex. With the availability of terabytes of RNA-Seq data and continuous development of analysis tools, there is a pressing requirement for frameworks that allow for fast and efficient development, modification, sharing and reuse of workflows. Scripting is often used, but it has many challenges and drawbacks. We have developed a python package, python RNA-Seq Pipeliner (pyrpipe) that enables straightforward development of flexible, reproducible and easy-to-debug computational pipelines purely in python, in an object-oriented manner. pyrpipe provides high level APIs to popular RNA-Seq tools. Pipelines can be customized by integrating new python code, third-party programs, or python libraries. Researchers can create checkpoints in the pipeline or integrate pyrpipe into a workflow management system, thus allowing execution on multiple computing environments. pyrpipe produces detailed analysis, and benchmark reports which can be shared or included in publications. pyrpipe is implemented in python and is compatible with python versions 3.6 and higher. All source code is available at https://github.com/urmi-21/pyrpipe; the package can be installed from the source or from PyPi (https://pypi.org/project/pyrpipe). Documentation is available on Read the Docs (http://pyrpipe.rtfd.io).

Introduction

Since its inception, RNA-Seq has become the most widely used method to quantify transcript levels (1); terabytes of publicly available RNA-Seq data, encompassing multiple species, organs, genotypes, and conditions, is deposited in public repositories (2). Integrated analysis of aggregations of thousands of diverse RNA-Seq samples enables exploration of changes in gene expression over time and across different conditions (3).

A major challenge of processing thousands of RNA-Seq datasets, is implementing data processing pipelines in an efficient, modular, and reproducible manner. Most bioinformatics tools are standalone linux programs, executed via the shell; bioinformatic pipelines are usually written as shell, perl, or python scripts, which may be integrated with Makefiles (4, 5). Scripting, although powerful and flexible, can be difficult to develop, understand, maintain, and debug.

Here we present pyrpipe, a lightweight python package for bioinformatics researchers to code and execute RNA-Seq workflows in an object oriented manner. pyrpipe provides high-level APIs for 15 popular RNA-Seq analysis tools including a dedicated module to easily access and manage RNA-Seq data available from the NCBI-SRA database (2). Researchers can integrate into pyrpipe their own python code, third-party programs, and existing python libraries in a flexible, straight-forward way. No new workflow syntax specific to pyrpipe are required. pyrpipe meticulously logs information related to pipeline execution, providing extensive debugging resources. After each analysis, researchers can generate reports and summaries with all information necessary to reproduce the analysis.

pyrpipe is not a workflow management system, such as the popular Snakemake (6) or NextFlow (7), in that it does not scale jobs on clusters, or manage memory and parallel processing. However, pyrpipe is designed to be readily integrated into workflow management systems, providing a customizable framework for reproducible and scaleable pipelines.

Implementation

We developed pyrpipe to create an easy-to-use python framework for researchers to code, share, and reuse RNA-Seq analysis workflows. pyrpipe achieves this by providing: 1. high level APIs to popular RNA-Seq tools; 2. a general API to execute within python any shell command, enabling use of any bioinformatics tool; and 3. extensive logging details of the commands. We selected the Python platform because it is widely used, free, flexible, object-oriented, and has high-level data structures (8), (9), with a repository of > 200,000 packages and tools.

A. Object-oriented and modular design. We have taken an object oriented approach to implement pyrpipe, such that any RNA-Seq processing workflow can be intuitively executed by the researcher. pyrpipe's modular design permits writing code that is easy to read, manage, and share. From a developer's perspective, modularity facilitates reuse and extensibility; new tools can be easily integrated into pyrpipe.

pyrpipe consists of highly cohesive modules (sra, mapping, alignment, quant, qc, tools) designed to capture steps integral to RNA-Seq analysis (Supplementary Table
**pyrpipe**

1. The modules **pyrpipe_engine**, **pyrpipe_module**, and **pyrpipe_diagnostic** contain multiple helper functions and classes.

Using classes to encapsulate various “tools” and “data” is the key concept in **pyrpipe**. For each module, we identified and implemented abstract classes to represent operations: access to NCBI-SRA, quality control, read alignment, transcript assembly and transcript quantification. To date, we have implemented 17 children classes providing APIs to specific RNA-Seq tools. (Supplementary Table 1 and Supplementary Fig. 1). Thus, “tools” can be easily accessed as objects, while ensuring that associated data and parameters are consistently accessible within that object.

A workflow is defined using instances or objects of these classes (Fig. 1). Once objects are created, they can be reused throughout the program, promoting faster development and code re-usability, for example, a Hisat2 object can be used to align reads from multiple SRA objects.

**B. Flexibility in pipeline execution, debugging, reproducible analysis, and pipeline sharing.** **pyrpipe** flexibility extends to choice of how the pipeline is designed to execute, and handle exceptions and errors. Researchers can create checkpoints in the pipeline, save the current **pyrpipe** session, and resume later. This is particularly useful for running different blocks of a workflow in different environments. For example, on a typical high performance computing (HPC) cluster, a researcher might use a dedicated data-transfer node to retrieve data from SRA and then use compute nodes for data processing.

**pyrpipe** has automatic logging features for efficient error detection and reports (Fig. 1). **pyrpipe_diagnostic.py** module includes a logger object and logs all executed commands and their outputs. Environment information, such as operating system and Python version, along with version and path information for each program used within the pipeline, are also logged. **pyrpipe** logs are saved in JavaScript Object Notation (JSON) format for easy parsing by other programs (Supplementary Table 2).

The **pyrpipe_diagnostic.py** module generates comprehensive reports about the analysis, benchmark comparisons, shell commands, reports for debugging, and MultiQC reports (10). These reports, along with the python scripts, can be shared or included with publications for reproducible research.

**Example usage**

**Transcript assembly using **pyrpipe**.** We demonstrate **pyrpipe**’s usability by processing *Zea mays* RNA-Seq data available through NCBI-SRA (2). The workflow is explained in following steps:

1. Importing **pyrpipe**: To use **pyrpipe**, we need to import it in current python session. Lines 1-5 imports the **pyrpipe** modules in python. Line 7 initializes a list of SRR accessions used in this examples. Line 10
initializes `workingDir` variable which contains the path where all data will be downloaded.

```python
from pyrpipe import sra
from pyrpipe import mapping
from pyrpipe import assembly
from pyrpipe import qc
from pyrpipe import tools

runs=['SRR3098746',
     'SRR3098745',
     'SRR3098744']
workingDir='maize_out'
```

2. Downloading raw data: First we created SRA objects corresponding to each SRR accession (Line 13). To download raw data from NCBI-SRA, we used the `download_fastq()` (line 14) to download reads in fastq format. We used Trim Galore object (line 19) and performed quality filtering (line 23). Parameters were passed to Trim Galore object as a dict (line 19).

```python
sraObs=[]
for x in runs:
    ob=sra.SRA(x,workingDir)
    if ob.download_fastq():
        sraObs.append(ob)

# create a TrimGalore object
tgOptions={"--cores": "10"
}
tg=qc.TrimGalore(**tgOpts)
for ob in sraObs:
    # perform qc using trim galore
    ob.perform_qc(tg)
```

3. Mapping reads and transcript assembly: We create an object to use STAR aligner (line 32) and StringTie (line 34). Then we process all SRA objects in a loop (lines 37-39). First, mapping the reads to the genome using STAR (line 38) and the performing transcript assembly with StringTie (line 39).

```python
starParams={'--outFilterType':
            'BySJout',
            '--runThreadN':
            '8',
            '--outSAMtype':
            'BAM SortedByCoordinate'
            }
star=mapping.Star(star_index='index',
                  **starParams)
st=assembly.Stringtie(reference_gtf= 'ref.gtf')
```

Case studies

C. Prediction of long non-coding RNAs (lncRNAs) in RNA-Seq Zea mays by supplementing pyrpipe with a third-party tool. We downloaded RNA-Seq data from SRA, quality filtered using Trim Galore (11), aligned reads to the Maize genome using STAR (12) and transcripts were assembled using StringTie (13). Then, we used a third party tool, (PLncPRO (14)), to predict lncRNAs, and assembled transcripts. Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Maize_lncRNA_prediction.

D. Arabidopsis thaliana transcript assembly using pyrpipe checkpoints. We downloaded raw read RNA-Seq data for Arabidopsis from SRA, performed quality control using BBduk (15), aligned reads to the genome using Hisat2 (16) and assembled transcripts using StringTie (13). Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Athaliana_transcript_assembly.

E. Integrating pyrpipe scripts within a workflow management system. We embedded pyrpipe into the Snakemake workflow management system (6), and used it to download human RNA-Seq data with SRAtools, quality filter the data with BBduk (15), align reads with Hisat2 (12), assemble transcripts with StringTie (13) and Cufflinks (17), and merge the multiple assemblies with Mikado (18). Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Human_annotation_snakemake.

F. Prediction of Zea mays orphan genes. In this case study we used ten diverse Zea mays RNA-Seq samples from NCBI-SRA to identify transcripts that would encode candidate species-specific (“orphan”) genes. Supplementary Fig. 2 shows the workflow, pyrpipe scripts, downstream analysis code, and data is available at https://github.com/lijing28101/maize_pyrpipe. The results are discussed below.

Results

The orphan genes of the current high-quality genome of Zea mays B73 had not been systematically annotated, thus, we examined the trends among orphan vs non-orphan transcripts of ten RNA-Seq runs from this line. Our analysis pipeline for this RNA-Seq data identified a total of 60,999 distinct transcripts that contained an ORF greater than 150 nt. A subset of these will represent protein-coding genes; others will be lncRNAs or expression products that do not represent genes.
6,306 of these transcripts contained ORFs whose translated product shows no homology to proteins of any other species ("orphan-coding transcripts"). Fig. 2 shows the transcript length, and GC content distribution for these orphan-coding and non-orphan-coding transcripts. The length of orphan-coding transcripts is shorter than for non-orphan-coding transcripts. However, GC content distribution is indistinguishable between orphans and non-orphans. These trends are quite similar to those of annotated orphan and non-orphan genes from Arabidopsis thaliana (19), although the median number of exons reported in orphan-coding transcripts in A. thaliana is one, versus that in Z. mays of two.

We compared the expression level of orphan and non-orphan transcripts within each RNA-Seq sample (Fig. 3). In each of the 10 runs analyzed, median expression of orphan-coding transcripts is much lower as compared to median expression of non-orphan-coding transcripts. However, in each run but one, some orphan-coding transcripts are highly expressed.

Conclusion

The pyrpipe package allows researchers to code and implement RNA-Seq workflows in an object oriented manner, purely using python. pyrpipe can be integrated into workflow management systems or used directly. Access to NCBI-SRA is automated, such that researchers can readily retrieve raw RNA-Seq data. The downloaded data and data files are automatically managed, and consistently accessed through SRA objects. Researcher need not keep track of data files or their paths, as these are integrated with pyrpipe objects. pyrpipe workflows can be modified using python’s control flow abilities and a user can create complex, reproducible, workflow structures. Any third party tool or script can be integrated into pyrpipe for additional data processing capability. pyrpipe logs and reports enable debugging and reproducibility. texttpyrpipe workflows provide a clear record for publications.

pyrpipe will appeal to researchers who are looking for simple, fast way to deploy large RNA-Seq processing pipelines. Straightforward implementation, execution and sharing of RNA-Seq workflows makes it an ideal choice for researchers with less computational expertise.

Funding

This work is funded in part by National Science Foundation grant IOS 1546858, Orphan Genes: An Untapped Genetic Reservoir of Novel Traits, and by the Center for Metabolic Biology, Iowa State University.

Bibliography

Fig. 2. Comparison of length, and %GC content among transcripts whose ORFs are orphan-coding and non-orphan proteins, as identified by our analysis pipeline.

Fig. 3. Box plots showing expression level (TPM) of expressed orphan and non-orphan protein coding transcripts in 10 RNA-Seq runs.
Supplementary Information.
Table 1. Currently implemented `pyrpipe` modules. Each module contains multiple classes containing APIs for different RNA-Seq tools.

Fig. 1. A UML class diagram showing `pyrpipe`’s classes and relationships among them. Classes in the same `pyrpipe` module share the same color.
Fig. 2. A flowchart showing the pipeline implemented in pyrpipe to identify potentially orphan coding transcripts in Zea mays.
### Table 2. Table showing description of the JSON keys stored in pyrpipe logs

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>shell command executed</td>
</tr>
<tr>
<td>starttime</td>
<td>Time at the start of execution</td>
</tr>
<tr>
<td>runtime</td>
<td>Total runtime</td>
</tr>
<tr>
<td>exitcode</td>
<td>The return code</td>
</tr>
<tr>
<td>stdout</td>
<td>stdout returned by the program</td>
</tr>
<tr>
<td>stderr</td>
<td>stderr returned by the program</td>
</tr>
<tr>
<td>objectid</td>
<td>Id of an object used with the command</td>
</tr>
<tr>
<td>commandname</td>
<td>Name of the command</td>
</tr>
<tr>
<td>python</td>
<td>Python version</td>
</tr>
<tr>
<td>os</td>
<td>Operating system</td>
</tr>
<tr>
<td>cpu</td>
<td>CPU information</td>
</tr>
<tr>
<td>syspath</td>
<td>Python’s sys.path</td>
</tr>
<tr>
<td>sysmodules</td>
<td>Python’s sys.modules</td>
</tr>
<tr>
<td>name</td>
<td>Name of the program executed</td>
</tr>
<tr>
<td>version</td>
<td>Version of the program</td>
</tr>
<tr>
<td>path</td>
<td>Path to the program on disk</td>
</tr>
</tbody>
</table>