DeNTAS:

De Novo Transcriptome Analysis & Statistics

MSc in Bioinformatics group software development project (2017)

Team Mandarin:
Dr. Dania Vicente-Zamarreno
Kristina Markova
James Nicholson
Yusef Badi

Supervisors
Professor Conrad Bessant
Dr Fabrizio Smeraldi

Queen Mary
University of London

1.1: DeNTAS: De Novo Transcriptome Analysis & Statistics

DeNTAS is a software tool designed to provide statistical analysis and visualisation of transcriptomic datasets generated by de novo assemblies. DeNTAS takes raw data in the form of unidentified assembled transcript FASTA sequences with their associated FPKM (Fragments Per Kilobase of transcript per Million mapped reads) values. DeNTAS leverages the computational power of Queen Mary's University of London's high performance cluster computer, Apocrita, to rapidly identify uploaded transcripts via a local BLASTn search. Multivariate data exploration and statistical inference and visualization of differential gene expression are carried out locally using EdgeR and Limma R. packages. Results are returned to the user, in the form of an interactive table of differentially expressed genes and downloadable principal component plots, dendrograms, volcano plots and heatmaps. DeNTAS is a prototype designed to demonstrate the potential of the software and currently only supports data from *Petropus Alecto*, *Homo Sapiens* and *Mus Musculus* and restricts the user to choose between two and four experimental groups for analysis. DeNTAS was created during the QMUL MSc in Bioinformatics group software development project 2017. The following documentation describes the structure and function of DeNTAS and details the motivations of our design choices and discusses both the current limitations of the software and opportunities for future development.

1.2: Acknowledgements

We would like to extend our gratitude to all who helped in completing this work including our supervisors Professor Conrad Bessant and Dr Fabrizio Smeraldi for their helpful guidance, appraisal, and constructive criticism of our work. Furthermore, a special thanks to Dr. Adrian Lärkeryd who helped set up the Apocrita access and a warm thanks to Nazrath Nawaz for sitting in on group meetings and offering valuable insight.

Table of Contents

2.0: DeNTAS software Design and Functionality

2.1: Software architecture

When developing DeNTAS we sought to produce software that coupled an attractive and intuitive front end with an efficient backend that is fast and flexible. DeNTAS conducts statistical analysis and graphical visualisation of *de novo* generated transcriptome datasets. Figure 1, below, provides an overview of the software's' architecture and the flow of information and files during analysis.

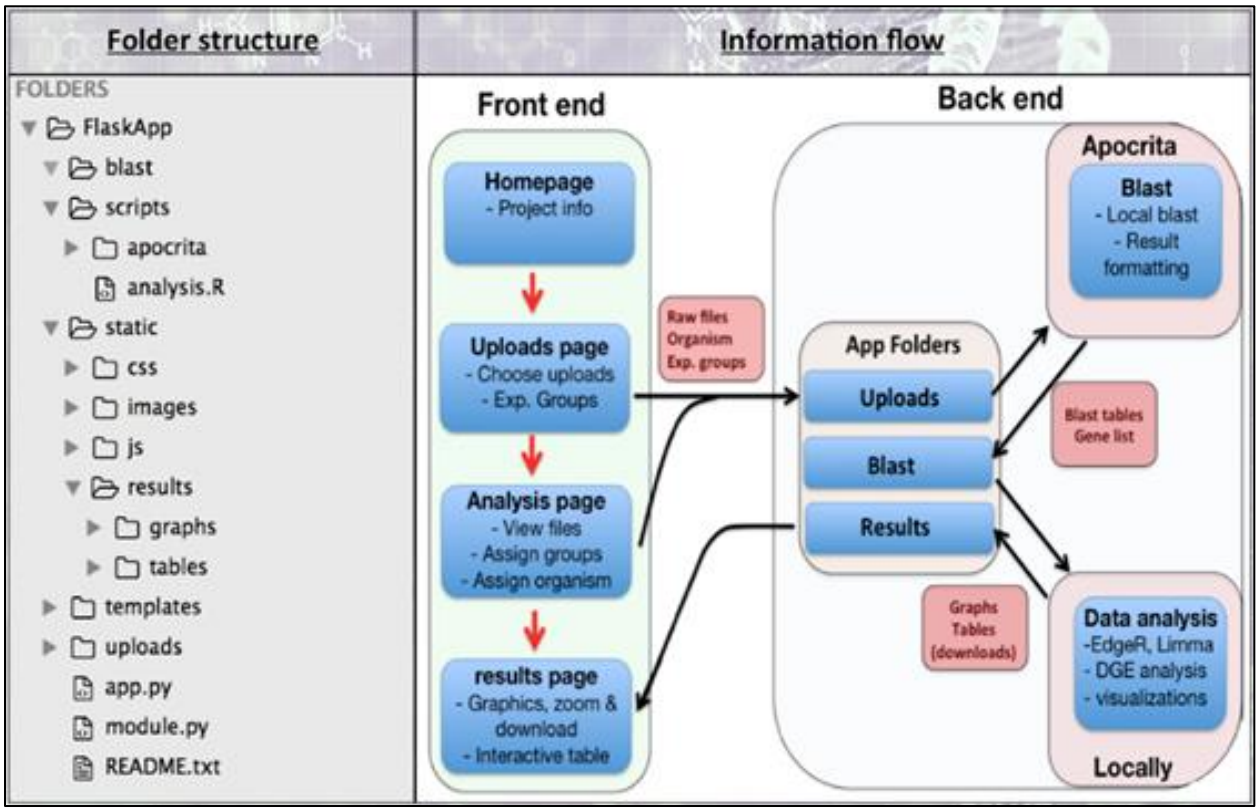

Figure 1. DeNTAS software is split into the Front end (graphical user interface) and a back end where both remote (blast search) and local (R statistics) processing take place. The user inputs their experimental parameters and the raw FASTA files are transferred to the uploads folder. From here files are sent for blast transcript identification and returned to the blast folder. The blast results are then analysed locally with tables and graphs saved in the results folder. Results are then rendered in the results page for visualization by the user.

The statistical analysis and graphical visualizations carried out by DeNTAS requires the following raw input and information: (i.) Raw FASTA files of unidentified assembled transcripts and their FPKM values, (ii.) The names of the different experimental groups (iii.) Assignation of each sample to a specific experimental group, (iv.) The choice of

organism specific database to blast the sample data against. In the front end, the user is asked to provide this information over two analysis pages (see Fig. 2). In the first page the user inputs the names of their experimental groups and chooses the files from their local machine to upload for analysis; these are then uploaded into DeNTAS' uploads folder. Next, the user is presented by a page with a dropdown box to choose the organism of their samples and a link to each uploaded sample file and a drop down box populated by their experimental groups. The organism field, defaults to *P. alecto*, corresponding to the sample dataset used for development, but *H. sapiens* and *M. musculus* are also supported (see section 3.2.1 for why these were selected). Once the user has selected an experimental group for each sample they can submit their data for analysis.

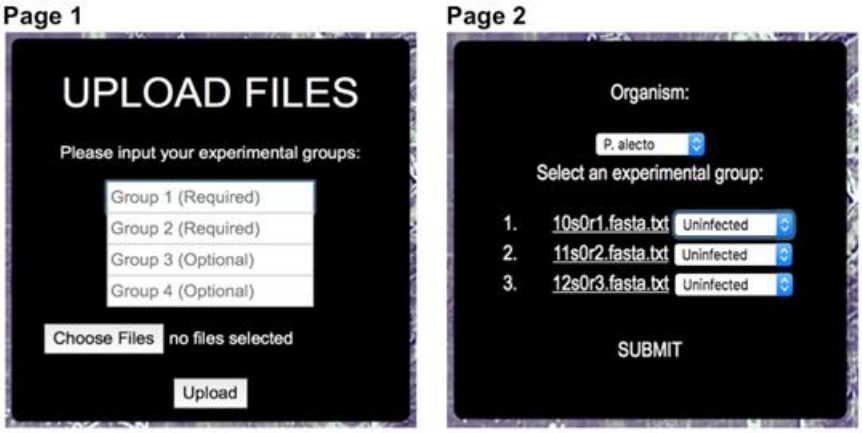

Figure 2. **(a.)** Page 1 where files input by the user are loaded; **(b.)** Page 2, where the user can select the organism of choice, and the experimental group.

Running a BLASTn local alignment on a full transcriptome dataset is a relatively time consuming computational process; we therefore chose to make use of Apocrita, QMUL's high performance cluster computer to significantly increase the speed at which DeNTAS is able to identify transcripts. We use secure shell's (ssh) secure copy (scp) function to upload the contents of the app's upload folder to Apocrita where a BLASTn search against an organism specific ref-seq mRNA local blast database is conducted.

The resulting tabular blast output files, along with a file containing the list of gene symbols corresponding to each ref-seq accession number are then downloaded from Apocrita and onto the local user's app's blast folder. The blast output files along with the experimental group information are then parsed into R for the statistical analysis of differential gene expression.

During analysis the following files are then exported from R into the app's results folder:

- The main data matrix of FPKM values (rows = genes, columns = samples).
- Multivariate data analysis graphs (PCA plots & dendrograms) for both the full gene set and the subset of genes that are differentially regulated.
- Volcano plots for each comparison between experimental groups.
- A heatmap of the top 100 differentially expressed genes across all group comparisons.
- A table of all the differentially regulated genes and their p-values.

Finally, the user is returned a results page (see Fig 3), which displays all of the graphs, complementary pop-up information boxes explaining each of the graphs, and an interactive results table showing the differentially regulated genes. All content is downloadable by the user. When using our full example dataset (n=9, file size ~70Mb) backend analysis typically takes <10 mins.

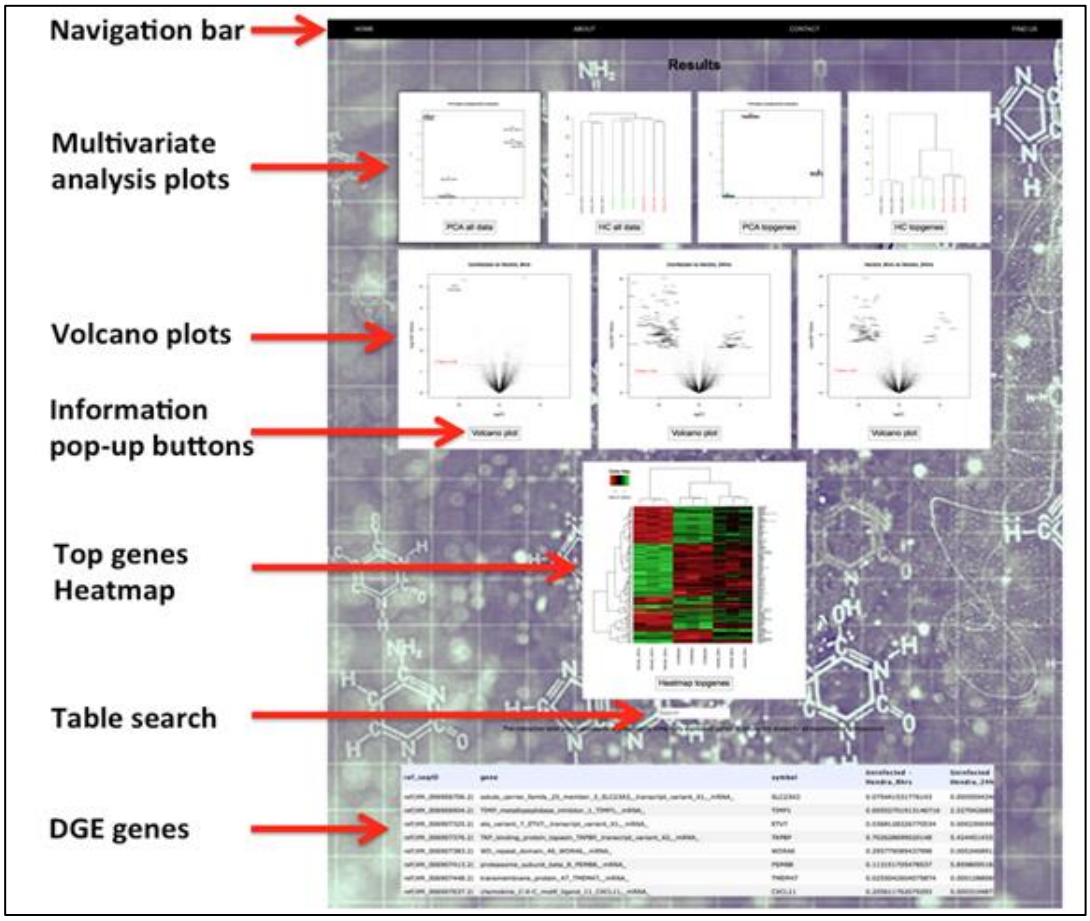

Figure 3. An annotated figure of displaying a typical results page returned to the user for an experiment consisting of three experimental groups.

2.2: App folder structure and contents

DeNTAS utilises a systematic folder structure to facilitate for the application's effective functioning. Folders are organised to enable the efficient hosting of the web-app on the local user's server (e.g. desktop computer) in conjunction with the streamlined direct accessing of the remote Apocrita server. Together this allows for the storing and retrieval of user-uploaded data, the generation of temporary files, and the final results generated output.

2.2.1: Local folder structure and contents

DeNTAS/

- **blast/** *this folder stores the tabular tsv files output by the blast script and downloaded from Apocrita.*
- **scripts/**
    - **apocrita/** *This folder contains all of the scripts and organism specific FASTA transcriptome databases necessary to set up a local computer to run DeNTAS. All scripts and files are hosted and executed remotely on Apocrita.*
    - **analysis.R:** *A soft coded R.script, which reads in blast results from DeNTAS/blast/, performs statistical analysis and outputs tables into DeNTAS/static/results.*
- **static**
    - **css/** *This folder contains the css file to format the csvToTable JQuery plugin generated dynamic results table.*
    - **images/** *This folder contains all the websites stock images used to populate the html templates.*
    - **js/** *This folder contains two JavaScript JQuery plugins; jquery.csvToTable.js & jquery.tablesorter.dev.js.*
    - **results/ graphs/** *This folder stores the tabular .tsv files output by the analysis.R script.*
        - **tables/** *This folder stores the figure png files output by the analysis.R script.*
- **templates/** *This folder stores DeNTAS' html templates.*
- **uploads/** *This folder stores the raw FASTA files uploaded by the user for analysis.*
- **app.py:** *This is the principal python script that is run to initialize the DeNTAS Flask app.*
- **module.py:** *This python script defines the functions that are called from within the app.py script.*

2.2.2: Remote folder structure and contents (Apocrita)

[<username>@frontend8.apocrita~]$

- **/data/home/<username>/** *bash scripts used to submit the blast job to the Apocrita queuing system are stored in the user's home directory.*
    - **<organism>_call_blast.sh** *This script is called remotely from the local machine over ssh and contains the full path name enabling the use of the qsub command and then calling the <organism>_run_blast.sh script.*
    - **<organism>_run_blast.sh** *This qsubmit script, loads the required blast+ module, specifies the resources requested from Apocrita, and calls the <organism>_blast.py.*
- **/data/scratch/<username>/group_project/** *Local blast databases and blast script are stored in the group_project folder of the user's scratch space.*
    - **<organism>_blast.py** *Organism specific python script which runs a local blast for each sample in the raw subfolder and reformats the tabular output saving results in the results subfolder.*
    - **<organism>.db.txt** *Organism specific blast database.*
    - **raw/** *This folder contains the raw FASTA files copied across to Apocrita from the app uploads folder.*
    - **temp/** *This folder contains the primary tabular output from the blast before reformatting.*
    - **results/** *This folder contains the re-formatted blast tsv files which are copied across to the in app blast folder for statistical analysis.*

3.0: Software component specifications

3.1 Flask (A Python microframework)

Flask was developed by Armin Ronacher and is based on the Werkzeug Web Server Gateway Interface (WSGI) toolkit and Jinja2 template engine. Both of these are Pocoo projects formed by an international group of Python professionals (Grinberg, 2014). Flask is a microframework that allows web designers and developers from across the world to have control over the web development process. It has a robust core that adds basic functionality to the web applications, and a large variety of optional plugins. This makes it unique in its flexibility and easy to use. Flask is suitable for all kinds of projects and is specifically used for prototyping (Dwyer, 2016).

3.1.1 The DeNTAS app.py script

The app.py script runs the application and launches the DeNTAS software. Our Flask framework makes extensive use of python imports, which are included at the start of the app.py script. It is important to choose a standard order for all the imports made in the script (Knupp, 2013). The imports used are listed below:

- **from flask import:**
  - **Flask:** *imports the Flask class.*
  - **Render_template:** *renders HTML templates.*
  - **Redirect**: *returns a response object and redirects the user to another target location.*
  - **Url_for:** *dynamically builds a URL specific function.*
  - **send_from _directory:** *passes files to the user.*
- **from werkzeug import:**
  - **secure_filename:** *uploads a file and displays it to the user.*
- **import module**: imports library of backend functions stored in the module.py
- **import subprocess:** *allows to spawn new processes.*
- **import os:** *allows using the operating system.*
- **import numpy:** *stands for short for Numerical Python (foundational package for scientific computing).*

Flask web applications use a routing system that allows users to navigate between desired web pages within the same website. The link between a Uniform Resource Locator (URL) and function is defined by a *route*. The most common way in Flask to define a route is through the app decorator. Decorators are Python feature that modify function behaviour. Our app.py script contains routes with decorator of the following standard format:

@app.route (‘ / ’)

def X():

module.Y()

return render_template (example.html)

Here, X() represents a view function and the response returned by the view function is the HTML documents. The information enclosed within the angle brackets is dynamic part so that any URL can be matched to this route. Our script makes use of functions, module.Y(), imported from module.py library to run back-end processes before the templates are rendered.Listed below are the routes and view functions used in the app.py script (Grinberg, 2014):

- **@app.route("/")**
  - **def main()**; *this is the main application (app) and will return the index page with information about the project*

- **@app.route('/about')**
  - **def about()**; *this returns the about page with more information about the project.*

- **@app.route('/choose', methods=['POST'])**
  - **def choose():** *this returns the first analysis page in which the user inputs the names of their experimental groups and chooses their raw data to upload.*

- **@app.route('/upload', methods=['POST'])**
  - **def upload():***this returns the second analysis page in which the user inputs their model orgaism and uses drop-down boxes to assign each sample to an experimental group.*
  -
- **@app.route('/results', methods=['POST'])**
  - **def analyze():** *this route starts the major backend analysis, uploads files to Apocrita, runs the blast & downloads results before local statistical analysis in R.*

- **@app.route('/uploads/<filename>')**
  - **def uploaded_file(filename=None):** *This route is expecting a parameter containing the name of a file. Then it will locate that file on the upload directory and show it on the browser*.

The app.py script is finished with a run method and that launches the development server. The ***__name__*** == *"name"* is used only when the script is executed directly. To configure the mode of operation there are a several options that can be passed on the app.run(). The debug argument is set as true *(debug=True)* to activate the debugger and reloader and facilitate the development process (Grinberg, 2014).

```
if __name__ == "__main__":
    app.run(debug=True)
```

3.1.2: Functions stored in Module.py

In order to keep our main app.py script as streamlined and readable as possible, we created a module which contains the library (list of defined functions) used for DeNTAS' backend analysis. Module.py is imported into the app.py script where the backend functions are called within the different app routes. Listed below are the functions contained in the module.py script:

- **apocrita_upload():** *This function uses secure copy (scp) to upload the contents from the local ./uploads folder to the remote /data/scratch/<username>/group_project/raw folder on Apocrita.*

- **apocrita_blast(organism):** *This function takes the user-selected organism as an argument and calls one of three different organism-specific scripts over shh to commence the local blast alignment on Apocrita for that species.*

- **apocrita_download():** *This function uses secure copy (scp) to download the contents of the remote folder /data/scratch/<username>/group_project/results on Apocrita to the ./blast folder on the local machine (user PC).*

- **pdf_combine():** *This function makes use of Ghost script (3rd party pdf manipulation software) to merge all of DeNTAS' graphical output into a single pdf for the user to easily download.*

- **R_analysis(groups):** *This function uses subprocess to initialize the R analysis script and parse in the required variables as a single string argument.*

- **clean_up()**: *This function empties the apps local blast and uploads folders as well as emptying the raw, temp and results folders on Apocrita.*

3.2: Basic local alignment search tool (BLAST)

BLAST is a dynamic programming algorithm used for the local alignment and identification of protein or nucleotide sequences. Query sequences are searched against a blast database and the identity of the reference matches are returned along with various statistical parameters describing the quality of the match, such as '%' identity and Expect value (e value).

Blast searches can either be conducted remotely using NCBI's dedicated servers or on a local machine provided the required software and databases are locally installed. A remote blast offers full access to NCBI's extensive databases with the ability to subset searches by organism. Theoretically, this would be an advantageous feature for DeNTAS. However, NCBI imposes limits to the frequency and size of search submissions from individual users. Hence, a local BLAST is the only option, however, an additional consideration is that local alignment of large FASTAs with multiple sequences, such as whole transcriptomic datasets, is computationally expensive and would take a long time on a standard desktop computer. We therefore decided to run a local blast on QMUL's high performance cluster computer; 'Apocrita'.

3.2.1: Optimising DeNTAS blast parameters

The software development specification requested that *P alecto* be used as a model species for web-app design. In order to enable maximal usability, in the confines of a proof-of-concept prototype, databases for *H sapiens* and *M musculus* were set up as these species cover a major proportion of scientific research studies. A local database was set up for each of these supported organisms by searching NCBI's ref-seq database for organism and mRNA values. An organism specific reference-FASTA was then downloaded and transferred to Apocrita, where a local blast database was set up using the following command:

```
>> makeblastdb -in <organism>_fasta.txt -parse_seqids -dbtype nucl
```

When designing our blast.py script, we used python loops to cycle through file and results names and imported the operating system (os) module allowing us to export bash BLAST commands to Apocrita's Linux operating system. DeNTAS uses blast+ version 2.5.0 and executes BLAST bash commands of the following example format:

```
>> blastn -- db P_Alecto.db.txt -- query 10s0r1.fasta -- outfmt "6 qseqid sseqid stitle" -- evalue 0.001
-- num_threads 4 -- max_target_seqs 1 -- out blast_10s0r1.fasta
```

3.2.2: BLAST+ arguments

- **BLASTn :** *This command initializes blast and specifies a nucleotide sequence alignment.*
- **- db database :** *This argument specifies the organism specific local blast database to be used for the alignment.*
- **- query <sample.fasta> :** *This argument specifies the sample transcriptome FASTA file and to be used as the query sequence for the blast search.*
- **- outfmt "6 qseqid sseqid stitle" :** *This argument specifies the format of the desired BLAST output. 'out format 6' is a tsv file with columns corresponding to; (i.) qseqid: query sequence ID (For DeNTAS this is the trinity assembly transcript identifier and the corresponding FPKM number; (ii.) sseqid: subject sequence id (the ref-seq accession number); and (iii.) stitle: subject title (the extended name of the subject sequence)*.
- **- evalue 0.001 :** *This argument specifies a E-value cut-off, BLAST hits with an E-value above this threshold will not be returned in the results.*
- **- num_threads 4 :** *This argument enables the blast command to be split and executed across 4 cores, which corresponds to the number of cores requested from Apocrita on qsub of the BLAST script.*
- **- max_target_seqs 1 :** *This argument dictates that only the single most significant subject sequence hit is returned for each query and was chosen to enable us to uniquely tag each query sequence with a single ref-seq accession number.*
- **- out <result.txt> :** *This argument specifies the name of the output blast.tsv file produced by the BLAST search.*

3.2.3: Formatting the output

Opting for a custom tabular blast output lets us easily extract our required data (qseqid, sseqid & stitle) without the need to parse the large XML files usually produced by BLAST, which contain additional information. Furthermore, the tabular format of the BLAST output mean that we could easily extract the FPKM values from the query sequence IDs and reformat the data to be compatible with our R analysis using bash's text manipulation tool *awk.*

During analysis, we often found that multiple query sequences would redundantly align to the same reference sequence within the database. We believe that this is likely due to the de novo assembly of splice isoforms absent in the reference database. This is an issue since our analysis pipeline involves attributing a single FPKM value to each transcript identified from the BLAST search. A standard Trinity *de novo* assembly analysis pipeline uses RSEM (RNA-Seq by Expectation-Maximization) to uniquely assign

multiple mapping reads during alignment based on a probabilistic model before calculation of FPKM (Haas et al., 2014; Li & Dewey, 2011). We therefore decided to sum the FPKM values from redundant query-reference matches, as each read contributes to a single FPKM value, and all of those reads are attributed to the identified reference sequence. This was also achieved using the text manipulation tool *awk.*

3.2.4: Apocrita - DeNTAS interface

All of the scripts and local blast databases required to run a local blast are stored and executed on Apocrita (see section 2.2). There is an organism specific version of each of the three scripts required to run the blast, and Flask calls a different blast option depending on the user-selected organism species. The secure shell copy command is used to transfer files between the local server and DeNTAS' remote folders on Apocrita. During analysis, DeNTAS uses a secure shell to run a short '*<organism>_call_blast.sh'* bash script on the Apocrita frontend machine which contains the explicit path to the *qsub* command (stored on Apocrita). This in turn calls a second submission script '*<organism>_run_blast.sh'* that details the requested resources for the qsub job. This qsub job is actually the running of the final '*<organism>_blast.py'* python script which carries out the blast search.

All 'qsub' jobs are prioritised based on server capacity and demand, jobs with greater reserve-demands are placed lower in the queue and take longer hence this produced optimisation challenges. Since our software's' functionality can scale with the remote server capacity, we sought to balance the increase in computation speed achieved by using multiple cores with the increased waiting times caused by greater reserve-specification demands. On consultation of the BLAST and Apocrita guides we opted to spread the job over 4 cores, requesting a default 1GB of RAM per core. With these setting we found that a job that runs our blast script on 9 ~70Mb FASTA files will usually queue for <5 mins and complete within 10mins of execution.

3.2.5: Setting up local-server access to Apocrita

Making use of Apocrita's superior computational power significantly increases the speed of DeNTAS' backend processing and transcript identification. However, connecting to Apocrita from within the DeNTAS application presented significant additional challenges during development. Using DeNTAS from a local machine therefore requires several set-

up steps. Users who wish to run the DeNTAS prototype from a local machine independently must:

1. Have an Apocrita account that can be securely accessed without the need for a password. This can be achieved by setting up a secure shell authentication key and uploading it to Apocrita:
    i. **>> ssh-keygen (hit enter 3 times)**
    ii. **>> ssh-copy-id <username>@login.hpc.qmul.ac.uk (input your Apocrita password)**
    iii. **>> ssh <username>@login.hpc.qmul.ac.uk (test connectivity)**

2. Make changes to the user specific Apocrita path names to match their Apocrita username. The following files with the DeNTAS app require changes:
    i. DeNTAS/module.py
    ii. DeNTAS/scripts/Apocrita/* (all 10 scripts in the Apocrita folder)

3. Open a terminal and navigate to the DeNTAS/scripts/Apocrita folder and then run the apocrita_setup.sh script. This script will:
    i. Use scp to upload the required scripts and compressed FASTA files to the user's Apocrita home directory.
    ii. run the call_blast_setup.sh script on Apocrita which will:
    iii. Create the required DeNTAS directories in the user's Apocrita scratch space
    iv. Move the required compressed FASTA files and scripts to the newly created directories and then uncompress the FASTA files
    v. Load the blast+ module and create the local blast databases for each organism.

3.3: Data Analysis in R

3.3.1: R version and required packages

We chose to conduct our data analysis in R for a number of reasons; As well as being one of the leading scientific tools for statistics and data visualization (reference), R is open source with a large and active community of users and an extensive list of biological statistics packages available on CRAN and Bioconductor. DeNTAS's statistical analysis is conducted in R version 3.3.1 (2016-06-21) and makes use of the Limma and EdgeR transcriptome analysis packages and the gplots and dendextend packages for graphical visualization of the data.

3.3.2: Data input

To enable the analysis of different user input datasets we use the python module subprocess to call the analysis.R script and pass in the required variable arguments;

```
>> subprocess.call(cmd, universal_newlines=True)
```

Where cmd corresponds to the command, *Rscript,* the path to the script, *scripts/analysis.R,* and args, a character string made up of; (i.) the list of files to be read into R and; (ii.) the experimental groups to which each sample corresponds. Within the analysis.R script these arguments are read in as a single variable using;

```
>> myArgs <- commandArgs(trailingOnly = TRUE)
```

The variable myArgs is then subset to extract the required information. The analysis.R scripts reads in two type of information, both as tab separated variable files. The readDGE function from the EdgeR package is used to simultaneously read the re-formated blast output for each sample from the apps /blast folder and create a DGEList object which contains the $counts raw FPKM data matrix for each gene per sample. Secondly, the '1_gene_list.txt' file is read in as a table and used throughout analysis to look up the abbreviated gene symbol corresponding to each ref-seq accession number.

3.3.3: Soft-coding strategies

The analysis.R script is soft coded to accommodate variable file numbers and names that are passed in from the app.py scripts. Furthermore, the data analysis can accommodate from 2 (mandatory minimum) to 4 (maximum) experimental groups. Implementation of a higher number of groups would have been a greater challenge, the majority of gene expression research has between two to four experimental groups (with few exceptions) hence this is the number we chose to accommodate for. the names of which are used to label graphs and experimental contrasts within Limma models. The number of experimental comparisons, and therefore the number of Volcano plots or columns in the tabular output of differentially expressed genes, are dictated by the number of experimental groups which is saved within the analysis.R script as the variable 'group_number'. Throughout the analysis.R script, the integer value group_number is used to dictate the choice in if else statements and points at which divergent analysis is required according to group number.

3.3.4: Statistics and visualizations

Raw FPKM values from the DGEList object are log2 transformed which addresses the issues of skewness and extreme values (high & low) in the data set *(Zwiener, Frisch, & Binder, 2014)* and is a prerequisite for *limma* analysis. Next, the data is filtered to exclude stochastic noise and low background levels of expression from analysis. Unsupervised multivariate data analysis is carried out on the filtered dataset and a plot of the first two principal components and a dendrogram explaining the relationship between the samples is generated. Later in the analysis pipeline, this multivariate data analysis is repeated on the subset of genes that are differentially expressed. Many recent statistical packages designed to identify differential gene expression (*EdgeR & DESeq*) start from raw counts. Here we chose to use the *limma* package for statistical analysis since it is capable of working with FPKM values. *Limma* operates on a data matrix of expression values (rows =genes, columns=samples) and creates a linear model for each row in the data. These linear models analyse entire experiments as an integrated whole which has the effect of sharing information between samples and increases the reliability of statistical conclusions drawn from low sample numbers (Ritchie et al. 2015). We also take advantage of *Limma's* parametric empirical Bayes procedures and incorporate a mean-variance trend into the global variance estimate, which accounts for the reduced reliability of low expression values (Sartor et al., 2006). After statistical analysis, Volcano plots are generated for each experimental comparison and those genes which have a log2 Fold change ≥ and an adjusted p-value of < 0.05 are highlighted in red and labelled with the gene symbol. Finally, the gplots package is used to plot a heatmap of the top 100 genes that are differentially expressed across all experimental groups.

3.3.5: Data output

All graphical output is exported as portable network graphic (png) files to DeNTAS' /static/results/graphs folder. In addition a tsv file of the (log2 - transformed) data matrix is stored in the /static/results/tables folder for further analysis by the user, and a table is created from merging the gene names and symbol of those differentially regulated genes with the p-values from *limma's* efit object saved in the same folder.

3.4: Web design and development

3.4.1: Functionality of the HTML documents

Each of the HTML documents contain a navigation menu which allows the user to easily navigate between web pages within the same website. It also contains hyperlinks to other external web pages such as DeNTAS GitHub repository which contains all the files associated with the development of the software.

1. **Index.html:** welcomes the user and contains a 'GET STARTED' button that takes the user to the next web page.
2. **About.html:** contains short paragraph describing the project mission.
3. **Choose_uploads.html:** allows the user to input the experimental groups as they fit (from 2 to 4). The user then can browse the files and upload them successfully.
4. **Upload.html:** uses a python 'for loop' to fill a form with the name of the samples and a dropdown box allowing the user to select their experimental group. The information from this form is then requested by Flask and fed into the data analysis.
5. **Results.html:** displays the R graphics output generated by the R analysis and table showing all significantly expressed genes generated by the BLAST analysis.

3.4.2: HyperText Mark-up Language (HTML)

HTML is the main markup language for describing the structure of every web page. HTML uses tags that characterize text elements and this serves as an instruction for the web browsers on how the document should appear. In the development of DeNTAS, the HTML documents were written and edited using both Adobe Dreamweaver CC and Sublime text editor. Adobe Dreamweaver CC is flexible web design and development tool for creating user-friendly websites. It is used from front-end developers and designers across the world to balance and manage web applications. What makes it unique and easy to use is its visual aids, auto-indentation, code colouring and resizable fonts that collectively reduce errors and make the code readable. In addition, Sublime text editor is used as this an efficient way to optimize the code quality. As previously mentioned, the HTML documents contain elements consisting of markup tags. Every tag is composed of angle brackets such as <html>, <head>, <body> etc. For each opening tag there is a corresponding closing tag that tells the web browser the completion of a command. For instance <html> and </html> tags. The slash (/) in the closing tag indicates the command is completed. The <!DOCTYPE> indicates the version of the HTML used in the documents. In between headings, text paragraphs, line breaks and much more can be added. External links to other web pages, Cascading Style Sheets (CSS) and JavaScript are implemented in the HTML documents. See Appendix 1 for full explanation of the different tags (Duckett, 2011). HTML code formatting was optimised using a an open source tool 'jsbeautifier' from MIT, stored on GitHub (Lielmanis, 2017).

3.4.3: Cascading Style Sheets (CSS)

CSS is concerned with the representation of the web pages it is implemented in the HTML documents; all fonts, background images, sizes and colours. The CSS code is contained within the <style> tags in the HTML documents. Web browsers use this information when opening HTML documents.CSS makes the documents much easier to maintain and give much better control over the layout of the web pages. The same CSS code can be applied across multiple web pages to add more flexibility. This makes it easier to maintain and keep the consistency across multiple documents (Duckett, 2011).

3.4.4: JavaScript and JQuery

JavaScript is dynamic and object-oriented programming language. Alongside HTML and CSS, JavaScript is supported by the majority of the web browsers. JavaScript is an untyped language, meaning that the variables are not type-specified. It enables executable content within the web pages to be included. It can include programs that interactively engage the user and dynamically control the HTML content. To introduce a user interactive element to the results page, we made use of JQuery - a “fast, small, and feature-rich JavaScript library”. JQuery, along with the two additional JQuery plugins, jquery.csvToTable and jquery.tablesorter.dev.js, are stored within DeNTAS’ ‘../static/js/’ subfolder. Links to the js scripts are incorporated into the header of each of our results page, allowing the ('#table').CSVToTable() and ('table').tablesorter() functions to be called within the results html page. This facilitates the dynamic rendering of the differential gene expression results table stored within DeNTAS’ ‘../static/results/tables’ folder. As an additional feature, we added a search bar and wrote a JavaScript function directly into the results html pages that enables searching of the results table and displays, only those rows found by the search. Also, a JavaScript alert popup box is generated to display information to the user about the R graphics output. To generate a JavaScript alert popup box, an ‘alert()’ function was introduced within <script> tags and a short paragraph describing each of the graphs was written within the empty brackets. A corresponding </script> tag was introduced for each respective <script> tag, telling the web browser that the command is completed (Duckett, 2011).

3.4.5: Web standards and compatibility

An HTML document could be created both correctly and incorrectly. Web browsers have the ability to deal with small mistakes and cover errors in the HTML code, therefore it is important to follow the HTML standards and test the page on different browsers. World Wide Web Consortium (W3C) offers validation service checks whether or not the created HTML page follows these standards. This service can be found at validator.w3.org. It

allows to the user to validate the HTML documents by one of the following: URL checking, uploading the file or pasting some of the HTML code (Cederholm, 2004).

4.0 Limitations and opportunities for future development

The current version of DeNTAS represents a working prototype that demonstrates the future potential of the software. There are therefore numerous limitations to the software and ample opportunities for further development. These limitations would need to be addressed to fully achieve our aim of producing software that couples an attractive and intuitive front end, with a fast and efficient backend analysis, providing multiple users with an easy-to-use and scientifically useful analytic tool.

4.1: Taking DeNTAS online

One of the most important next steps in development would be to make DeNTAS accessible through a dedicated online-host and capable of handling multiple user interactions concurrently. In its current form DeNTAS can only be downloaded from the GitHub release and run locally, furthermore, there are several requirements and manual set-up steps needed before DeNTAS can be run on a new device (see section 4.6). Whilst using Apocrita enabled us to conduct large computations quickly in an application run from a desktop computer, it came at a significant cost, requiring substantial set-up and increasing app complexity. For instance, full pathnames must be used and functions explicitly called: it takes three separate scripts (call_blast.sh, run_blast.sh & blast.py) to run each blast analysis. Furthermore, separating our software back-end over two machines increases the challenge of introducing multiple user functionality. We investigated the possibility of using Flask's sessions library to either assign unique user-Ids to the files generated by a specific user, or creating temporary user-specific folders. However, using this approach would require the parsing of additional user-specific information generated locally to Apocrita, and either the altering filenames within scripts or changing folder structures remotely. We also considered introducing a queuing system workaround, through implementing an array on Apocrita and letting jobs run in order of submission. Ultimately this proved too great a challenge to overcome within the time constraints for development.

Launching DeNTAS as a functional webapp would therefore require a change in the architecture of the software. This could be done by making DeNTAS accessible online and hosting the application on an Amazon Web Services virtual machine which also conducts all of the backend processing. Hosting the app and conducting analysis on a single machine would simplify introducing multi-user functionality. Furthermore, since

DeNTAS' analysis is of a highly 'splittable' nature, a higher level of multi-threading could be introduced increasing DeNTAS' speed by having more threads reserved and utilised on-demand without queuing. Introducing multi-user functionality would also involve the creation of an account registration and login system with users' information securely stored in a database.

4.2: A more intuitive and responsive front end

A great front end should look professional, be clear and simple to use and maximize responsiveness to the user. With this in mind there are several improvements that could be made to improve the user experience of our software. Firstly, it is important to consider the variety of devices used to access the Internet, thus creating a website that is compatible with any screen size is important step. Websites that responsive and are able to rearrange and resize elements so that users from across the world from various devices would be able to access and browse with ease. Some of the technologies that could be used to design a responsive website include HTML5, CSS3, HTML5Shiv for Internet Explorer, media queries, photoshop image optimization and slicing and design adaptation for various screens (Frain, 2012). Secondly, another important feature, that we would have liked to include is a loading page or gif image that is presented to the user when navigating between pages. This is especially important for DeNTAS since it is an application that conducts relatively large and slow back-end processing tasks whilst navigating between pages. During further development this could be achieved by using AJAX requests to trigger the hiding and showing of different document elements within the html template. Finally, there are several improvements to our results pages that we would have liked to include given more time. For example, the details of the graphs displayed are quite difficult to make out, and we could have perhaps instead displayed a lower resolution thumbnail images and enabled a zoom magnification on hovering of the cursor. There are a large number and variety of JQuery plugins designed to offer this type of user interactivity that could be explored during further development.

4.3: Additional features

When designing our prototype we drew up a list of essential software features and functionality to meet the software specifications as well as a list of non-essential features that would be beneficial to the user. The majority of the non-essential features that we would add in further development involve either an increased optionality offered to the user, or some additional information returned in the results section.

It would be nice to return a detailed log of the analysis to the user that would include information such as the number of identified (& non-identified) transcripts per sample, and

the number of transcripts excluded from analysis during data filtering. It could also be important if DeNTAS included the expression of transcripts that, although not identified by blast, were present in multiple samples. For example, a multiple alignment could be done for the unidentified transcripts from each sample, adding an arbitrary reference tag and including the corresponding FPKM values in the downstream differential gene expression analysis. An important part of the biological interpretation of transcriptome studies is pathway analysis and/or gene enrichment analysis, this could explored in R using the topGO package that provides tools for testing gene ontology.

Finally, it would be advantageous to introduce further options to allow the user to customize their data analysis pipeline and to increase the number of organisms and experimental designs supported by DeNTAS. Some of the analysis parameters that we envision the user wanting to alter are: the Blast search E-value threshold; which R graphics to return; FPKM filtering thresholds; the number of differentially expressed genes displayed in the heatmap. These user choices could be parsed into functions as arguments and then stored as variables in python/R scripts.

4.4: Summary

In summary, DeNTAS is innovative piece of software and purpose built web- app tool designed for high quality *de novo* sequence assembly and statistical analysis. Despite the limitations of making a more responsive web design, fast and efficient back-end analysis, multiple users functionality it still functions as a viable proof-of-concept and is greatly promising. In many instances, current technologies are sufficient to perform de novo assemblies efficiently, however, from a research perspective DeNTAS provides focus on real-world bioinformatics problems. The increased progress of software engineering technologies such as the use of object-orientated, real-time operating systems greatly contributes towards future software development for this project (Dashchenko, 2003).

5.0: References

Cederholm, D.(2004). Web standards solutions: The Markup and Style Handbook. New York:Apress.
Dashchenko, A.(2003). Manufacturing Technologies for Machines of the Future: 21st Century Technologies ; with 41 Tables. Berlin: Springer Science & Business Media

Ducket,J. (2011). HTML and CSS: Design and Build Websites. Canada: John Wiley & Sons,Inc.
Dwyer,G.(2016). Flask by Example.Unleash the full potential of the Flask web framework by creating simple yet powerful web applicatio. Birmingham: Packt Publishing Ltd.

Frain,B.(2012). Responsive Web Design with HTML5 and CSS3. Birmingham:Packt Publishing.

Grinberg, Miguel. "The Flask Mega-Tutorial, Part I: Hello, World! - Miguelgrinberg.Com". Blog.miguelgrinberg.com. N.p., 2017. Web. 14 Feb. 2017.

Grinberg, Miguel. Flask Web Development. 1st ed. Sebastopol, CA: O'Reilly & Associates, 2014. Print.

Haas, B. J., Papanicolaou, A., Yassour, M., Grabherr, M., Philip, D., Bowden, J., … Pochet, N. (2014). De novo transcript sequence reconstruction from RNA-Seq: reference generation and analysis with Trinity. Nat Protocols (Vol. 8). http://doi.org/10.1038/nprot.2013.084.De

Knupp, J.(2013). Writing Idiomatic Python 3.3.

Li, B., & Dewey, C. N. (2011). RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC Bioinformatics, 12(1), 323. http://doi.org/10.1186/1471-2105-12-323

Lielmanis, E., et al. 'JSBeautifier', (2017), MIT. GitHub repository: https://github.com/beautify-web/js-beautify
Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., & Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Research, 43(7), e47. http://doi.org/10.1093/nar/gkv007
Sartor, M. a, Tomlinson, C. R., Wesselkamper, S. C., Sivaganesan, S., Leikauf, G. D., & Medvedovic, M. (2006). Intensity-based hierarchical Bayes method improves testing for differentially expressed genes in microarray experiments. BMC Bioinformatics, 7, 538. http://doi.org/10.1186/1471-2105-7-538
Zwiener, I., Frisch, B., & Binder, H. (2014). Transforming RNA-Seq data to improve the performance of prognostic gene signatures. PLoS ONE, 9(1), 1–13. http://doi.org/10.1371/journal.pone.0085150

6.0 Appendix

6.1: HMTL tags and formatting

Commonly used <head> tags:

- <title> Defines the title of the webpage.
- <meta> Defines any metadata used to specify page description.
- <style> Defines the style of the webpage in accordance with CSS stylesheet.
- <link> Links with external resources.
- <script> Introduces JavaScript code or link to external JavaScript file.

Commonly used <body> tags:

- <a> Links with different pages within the website but also to other external websites.
-
 Defines a line break.
- Defines a section.
- <footer> Defines copyright information.
- <form> Defines user input such as test field and many more.
- <h1> to <h6> Defines a number of heading level.
- <header> Defines container with introductory content.
- <img /> Introduces images to the webpages, src indicates filenames and its position and alt indicates the name of the image.
- <input> Defines a field that allows the user to enter data.
- Defines a 'list term'
- <p> Defines a paragraph of text.
- <select> Defines a list of options <option>
- Defines important text.
- <tbody> Defines a table.
- <td> Defines a normal table column.
- <! -- -- > Add comments.

6.2: CSS tags and formatting

body:

- font- family-specifies the font for an element.
- background-image-sets background image for an element.
- background-size- specify the size of the image.
- background colour-sets the background colour.