Annow: BLAST Based Analytical Sequence Annotation Software

Annow: BLAST Based Analytical Sequence Annotation Software

Ashkan Bigdeli

A Thesis in the Field of Biotechnology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November 2016

Abstract

As gene sequencing becomes more common the annotations that give meaning to biological data change. The large quantity of –omics data generated by these projects maintain static linkages to annotations that change as experimental evidence accumulates. The validity of data across several biological disciplines relies on the accuracy of these static linkages. This raises an informatics challenge as data sets large and small require diligent curation costing valuable resources. Annow was developed to be agile software to meet this challenge; updating data sets, while providing relevant alignment data, analysis metrics, and resource updating. Incredibly user friendly and operating offline Annow brings large data analysis capabilities to the personal computer and can update thousands of annotations in addition to more granular analysis in short time frames that are not possible with web-based solutions. The result of which is more a cost effective manner to produce reliable, accurate results applicable to any field utilizing biological data and annotations derived from gene sequencing.

Dedication

To those affiliated with the Dana Farber Harvard Cancer Center and the Harvard Community for fostering my career in informatics.

Acknowledgements

I would like thank Li Chan for actively encouraging me to pursue bioinformatics, the professors at the Harvard University Extension School for providing me the tools necessary to follow through on that pursuit, and to the Harvard Medical School community who turned my enthusiasm into ability.

I would also like to thank my family and friends, who accepted and supported someone focused on both work and school unconditionally. In particular I would like to thank my older brother, Afsheen Bigdeli, who to this day somehow answers all of my computer science questions and has provided invaluable guidance as I traded my pipette for lines of code.

Table of Contents

List of Tables

List of Figures and Graphs

Chapter 1

Introduction

The general study of –omics, primarily considered to be genomics, proteomics and metabolomics, has an impact that exceeds the field in both research and clinical settings. The increased understanding of genes and gene models has revolutionized biology and enabled a greater potential to shape the fields within. –Omics helps us better understand cancers, reproduction and disease, unlocking the potential to change medicine, agriculture, ecology, and even organisms themselves; the effect of which can be seen in a broad range from populations to individuals.

To effectively utilize –omics information, biological molecules must be linked to identifiers referred to as annotations. Annotations provide information on biological and biochemical function as well on gene regulation and interactions. As such the accuracy of the gene annotation mechanism becomes critical and maintaining this accuracy is a constant challenge. Inaccuracies arise from a fundamental limitation in –omics: only a small minority of genes in any sequenced genome have been characterized in experiments and the vast majority of annotations of other genes and proteins are inferred through these experiments on sequence similarity alone (NCBI, 2013).

As sequencing and experimental data accumulates, inferred annotations may shift to be match experimental evidence. The primary challenge then becomes updating static annotations and sequences to match the latest information. The pace of sequencing output suggests a method should be available to group and individual researchers so that dynamic annotations are easily achieved using fewer resources. It is the goal of the outlined work herein to provide an efficient, accurate and easy to operate method of overcoming this challenge. Annow, a program to update gene annotations, can be used on a large or small scale to service biologists and clinicians across fields.

Gene Sequencing

The catalyst for the -omics revolution has been, and continues to be, gene sequencing. The ability to sequence genetically meaningful information has gone through many iterations. The first iteration resembling modern day uses is considered to be Sanger sequencing, developed in 1977 and still used today. This technology allowed for the reading of relatively small genetic sequences utilizing the chain-termination (Sanger, 1977). This meant that genetic sequences at the base level could be now easily interpreted and correlated to biological observations.

In order to utilize available technology to cover larger sequences scientists developed a technique commonly referred to as "shotgun-sequencing". Shotgun-sequencing is a process where randomly-fragmented DNA is cloned into phage-vectors thereby creating libraries for random clone selection. The fragment containing clones then serve as a template for DNA sequencing (Anderson, 1981). Randomly-fragmented regions of DNA create overlapping sequences which could then be used to assemble long contiguous sequences (contig) by previously established computational methods (Staden, 1979).

By 1987 automated capillary instruments were introduced allowing this computational biology approach to be conducted at higher throughput. These high throughput machines were able to analyze several sequences at a time producing reads that reached up to 1 kilobase (Kb) and through contig assembly allowed for analysis of larger genomic regions. This “first-generation” approach allowed for projects of a larger scale such as the National Institute of Health (NIH) led Human Genome Project which resulted in the sequencing of the first human genome, an initiative that launched in 1990 and was declared complete in 2003 (Lander, 2001). The sequencing of the human genome led to many of the reference sequences utilized today to identify genetic alterations.

The technology used in the Human Genome Project was able to sequence 84 Kb per run at capacity and took 15 years and 3 billion dollars to complete. With the Human Genome containing over 3 billion base pairs the need for increased capacity in gene sequencing was clear. This lead to the introduction of a new method of sequencing in 2005 often referred to as “next-generation sequencing” (NGS). Reading many short reads in parallel on a massive scale, NGS increased capacity from 84 Kb to 1 gigabase (Gb) per run (Heather, 2016). What once took years, may now take a day. In fact in 2015 an entire human genome was sequenced for an actionable, clinical diagnosis in as little as 26 hours (Miller, 2015).

Impact of Technological Advances

As the technology for sequencing has advanced, the cost per base has decreased, making more projects and objectives possible. In contrast to the Human Genome Project, today a single NGS machine is capable of over 45 human genomes a day at roughly $1000.00 dollars a genome (Figure 1). This makes population, personalized medicine and gene model studies much more possible in many more labs and hospitals. The data generated, especially transcriptome data, directly effects the representation of gene models. Previously established locations for introns, exons, gene boundaries and start and stop loci of transcription can all shift as cumulative data helps establish a more accurate gene representation. While the wealth of information provided by these technologies is clear, the ability to apply this information faces two major challenges. The management of data generated, and the correlation of this data to biological meaning.

Independent of the technology employed, sequencing generates massive amounts of data. Moore's law is often projected to Biocomputing and when applied states that the amount of data generated will double yearly. Illumina™, a leader in NGS, shows that sequencing has significantly passed this threshold (Figure 1). With so much data generation, storage of only relevant sequences as it pertains to an objective is essential for most groups in terms of cost management. In practice this means most applications will involve storing data generated by sequencers, and reaching to an outside source for validation or reference comparisons. An example of this is the National Center for Biotechnology Information's (NCBI) BLAST tool. A commonly used informatics tool to align sequences against references (Altschul, 1990).

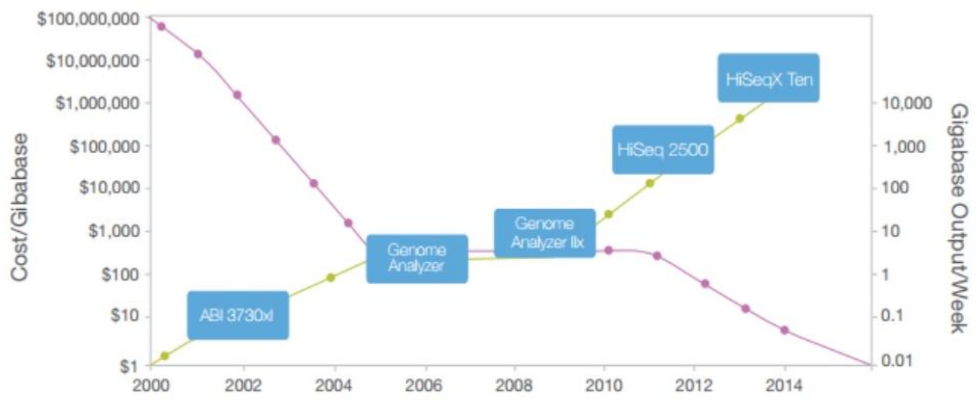


Figure 1. Illumina representation of cost per Gb and output over time (Illumina, 2016).

Tools like NCBI's BLAST assist –omics in a two-fold manner. A dynamically stored reference is provided and sequences will then be matched with an annotation. Annotations vary, not only by groups curating data, but within the datasets themselves. A given gene sequence can have several annotations matched depending on the application and annotation provider. To further complicate this data enrichment, annotations necessarily change over time so that their attachment to an accurate reference is maintained.

Utilization of Reference Sequences

Indeed the progression of the field of –omics hinges on the fidelity of information that is used as reference. As large-scale DNA libraries and sequence databases become more common it is important that the genes contained therein are accurately represented. The recent affordability of sequencing, the increased use of GWA (genome wide association) studies, and independent research frequently changes the scientific community's consensus representation of a gene. This can include refinement of introns, exons, splice sites and other relevant regions. It is for this reason the most popular resources for gene sequences provide frequent updates to their databases.

NCBI has become a common resource regarding the latest genetic information for several species from human to bacteria. Two initiatives are in place that allow researchers to utilize the latest reference genomes. The first, and most widely utilized is NCBI's Reference Sequence (RefSeq) project. The Reference Sequence collection is a comprehensive database containing the latest consensus for genomic sequences, transcripts, and proteins. This downloadable database provides the latest non-redundant annotation framework for research in expressive, comparative and gene characterization studies (NCBI, 2002). The second resource NCBI provides is the Consensus Coding Sequence (CCDS) project, a branch of NCBI that aims to identify core genes in human and mouse species responsible for protein coding (Pruitt, 2009).

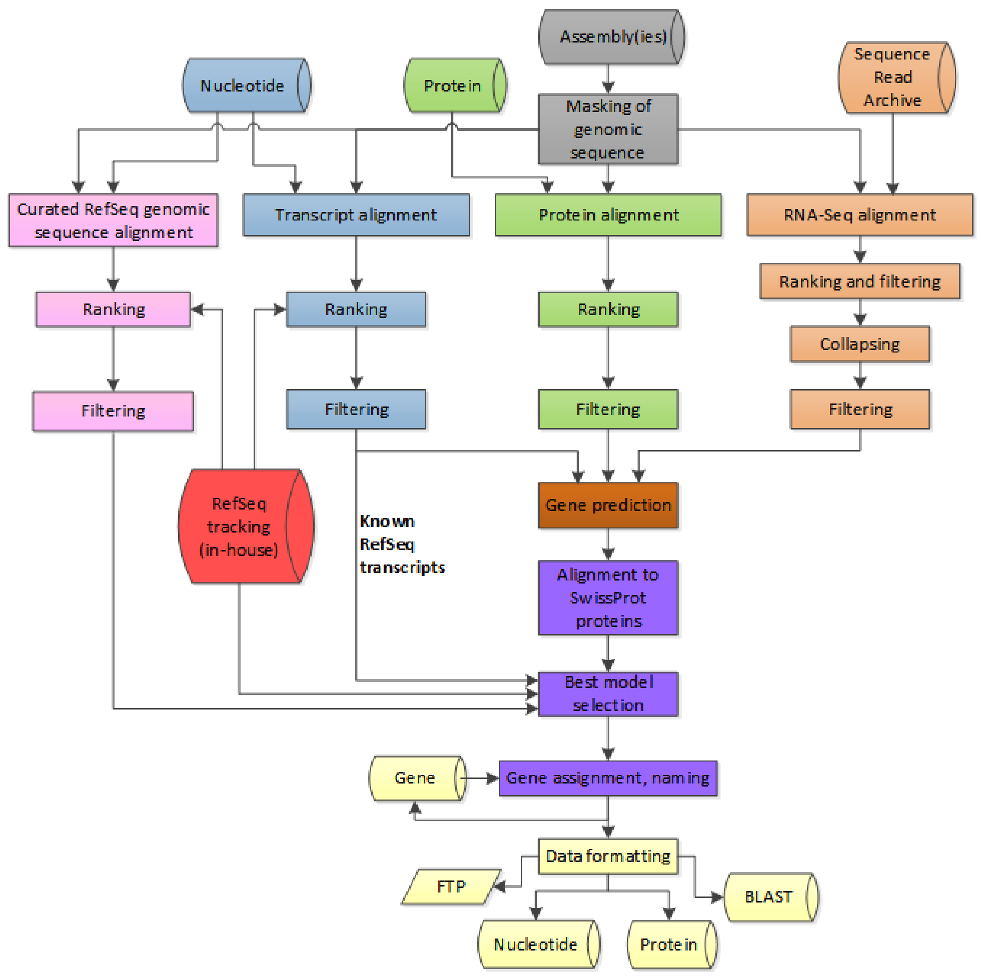


Figure 2. NCBI's annotation pipeline to validate and annotate sequences shows how many different resources are pooled to develop a single consensus sequence and corresponding annotation (NCBI, 2016).

NCBI's annotations to gene sequences are generated by compiling and curating internal sequencing projects and independent sequence submissions from the global scientific community. These sequence are then run through an in-house annotation pipeline prior to publication (Figure 2). Upon intake sequences are aligned to previous RefSeq entries and ranked based upon likely hood. This ranking system scrutinizes reference sequences, and therefore their annotations. Discrepancies can lead to a suppression, replacement, or removal of a given reference sequence or annotation. It is this process, which every annotation provider undertakes in some manner that brings the necessity for a mechanism to update previously recorded gene annotations.

An action of genome annotation was undertaken by NCBI's European counterpart, the European Institute of Bioinformatics (EIP). The goal of the EIP was to collaborate with genome sequencing groups to provide a frequently annotated library of genomes distributed through a web portal called Ensembl (Zerbino, 2015). While the EIP's initiative and aim is nearly identical to NCBI's, it sought to pool more institutions and research groups to compile consensus sequence releases. There are several other annotation mechanisms beyond NCBI and Ensembl, but each serve as a mechanism for – omics researchers to reliably pull information regarding their gene, or genome, of interest from a centralized database.

RefSeq Meta-Analysis

An analysis of overall fluctuation of NCBI's RefSeq annotations illuminates the need for an efficient, automated, annotation mechanism. All RefSeq annotations spanning all available organisms from a period of 2008 to 2015 were retrieved from NCBI's FTP change log and were clustered based upon record manipulation.

An attempt was made to retrieve species specific previous RefSeq releases, but as the releases are rather large, only the current release is available for download. As such only changes in yearly time points across all organisms was able to be mined effectively.

A tool like Annow can now be used to complete the change in species specific releases efficiently overtime, but presently only a broad overview of changes across all organisms and all releases is possible.

In analysis of NCBI's ReSeq change three types of record manipulations were observed. Removed records, referred to in change logs as "permanently suppressed", which are removed from the collection entirely. Records under review, referred to as "temporarily suppressed", which may later shift to permanently suppressed or replaced. And lastly, replaced values which are instances when an annotation is given a new identifier, or in rare cases, merged or linked to an existing identifier.

In total over 11 million records were manipulated in some fashion over a 7 year span, with the largest contributor being removed records. Over a 7 years span 6,523,212 ReSeq records were removed across 6 domains with a significant trend upward in 2015 (Figure 3). This data considered in the context of decreased sequencing cost at higher throughput is logical. However, so much of this data has been generated within the past 12 months it is hard to say if a year like 2015, in which 3,309,116 records were removed, is result of technological advances or a onetime curation measurement. Preliminary data of the same assessment for 2016 (not shown) showed a continuation of the upward trend, so it is reasonable to think that with more sequencers in more labs this upward tick is closer to the norm than an aberration.

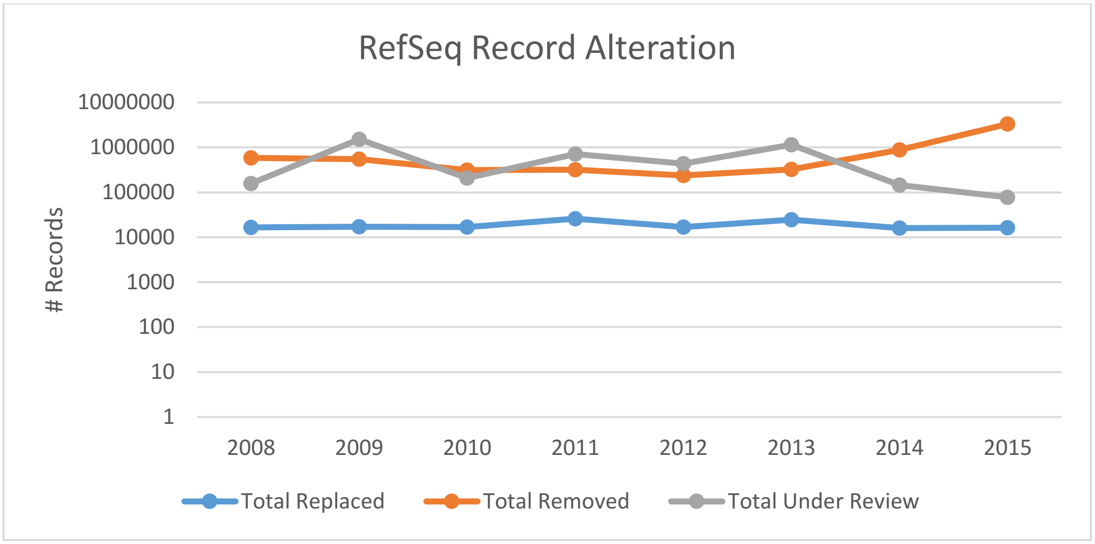


Figure 3. Logarithmic display of record manipulation over time shows replacements remain steady, while removals and reviews trend in opposing directions.

One would suspect that with such a large number of records removed, many would also be replaced. A replacement occurs in the event that contrary evidence is shown, and as shown previously, more data being generated presents a greater opportunity to affirm or refute existing genetic models. However, what we see is that replacement records remain fairly consistent over the 7 years span and are a relatively small number comparative to the whole data set comprising only 1.3 % of the total. This means that perhaps an annotation mechanism's greatest feature would be identifying sequences that are no longer relevant rather than those that require updating.

The above data was subset into only predicted coding and non-coding RNA annotations. This was done to try and use NCBI's predicted gene model, which takes into account a variety of datasets to determine the most likely transcript. While data is theoretical, this was thought to be more stable and therefore more representative of an overall trend in gene information.

When we look at this data again we see similar trends on a smaller scale. Predicted RNA RefSeq records across 6 domains contained a total of 1,951,871 annotation manipulations. Of this nearly 2 million; 1,729,525 or 88.6 % of total annotation change resided in removal. Only 64,110, or 3.2 %, records were replaced (Figure 4).

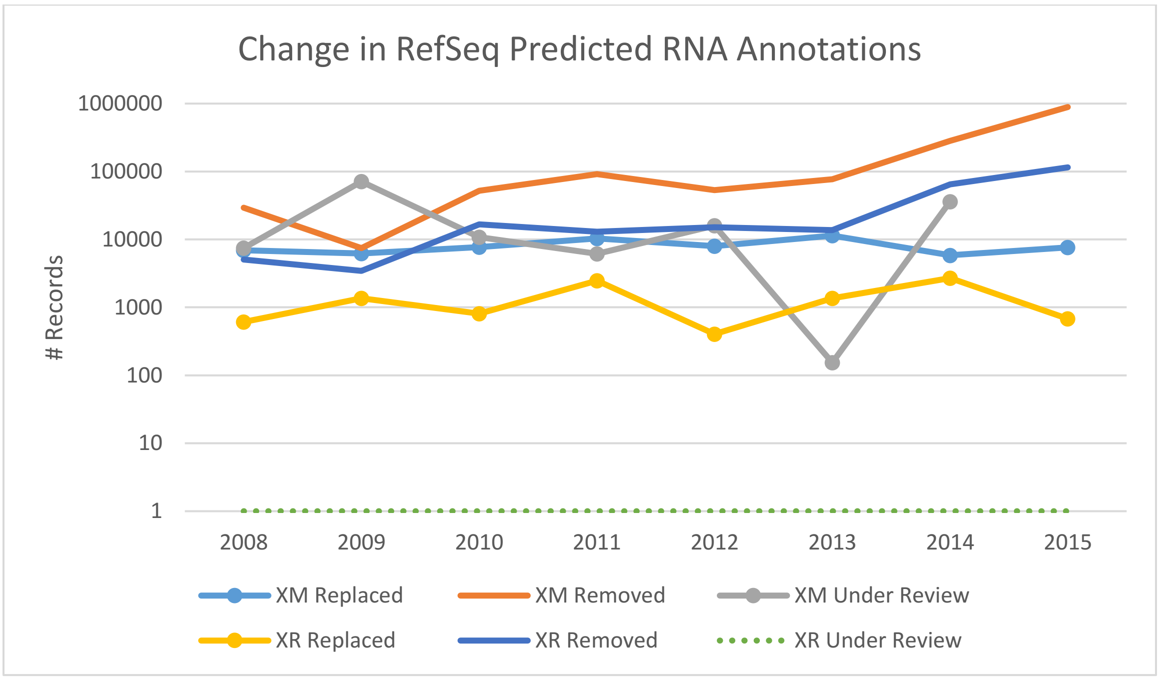


Figure 4. Logarithmic display of record manipulation over time shows replacements removals as the largest factor in record change. XR Under Review data showed no result.

There is also the nature of a record being “Under Review”, or temporarily suppressed. An attempt was made to correlate reviews to removal or replacement to identify what is the most likely course of action post review, but no trend was identified. This suggest an internal mechanism at NCBI (i.e, availability, curation) that causes the relationship of removal or replacement to be in at least some fashion separate from the review process.

With so many record manipulations the question then becomes, how can we utilize this information in an efficient, high impact manner to cross-reference our own records, experiments, or databases to ensure we have the latest consensus data and maintain concordance with our reference and annotations of choice?

Turning Static Annotations Dynamic

Annotating existing genetic data sets so that they reflect current genome information is a problem that many who work in the –omics field face and is often solved through custom one-off solutions that can be both costly, and time consuming. You may find an annotation tool on a research group’s website and typically these are low throughput tools specific to the group’s objective. Several open source annotation tools are also available, but focus mainly on specific gene alterations leaving a broader of view of genes absent. It is Annow’s aim to provide a mechanism to automate much of the process of in-house gene annotation and make the alignment with consensus gene data easily attainable whether the research involves a thousand records or several hundred thousand. The application of which can be utilized with NCBI's, Ensembl's, or other resources available information and applied to data types such as amino acid, nucleotide or ribonucleotide sequences.

This project provides researchers around the globe with a free, downloadable program with which to compare local databases to a NCBI, Ensembl, or a reference database of their choice in a highly efficient manner. Users may find this tool most useful for curating large sequence databases yielding more accurate starting materials and references and minimizing costly hours spent doing sequence analysis. This tool will also prove useful for those with query heavy projects that run over a longer period of time than is possible on the web.

The need for this arises because online sources such as NBCI and Ensembl implement query restrictions to prevent any one user from making too many requests and possibly overloading the system. For example, NCBI imposes a queue system when a user is interacting with their web interface. A simple request of five sequences will take no less than four minutes. This is done so that all users may use the shared resource without bogging down the framework on which it runs. When a researcher chooses to build a custom solution that interacts with NCBI's web portal in an XML or REST manner these limits are increased to three URL pings per second, and it is requested that large jobs be run during non-peak hours 9PM to 5AM EST (Sayers, 2010). Surpassing these limits will result in a ban for the user making the request.

By bringing the database off the web and into the local environment we eliminate web based limits to the number of queries performed on a database and eliminate an obstacle common to large scale analysis. Further, combining an offline approach with

NCBI's available local tools means that we can achieve higher query limits while maintaining the same algorithms commonly used in analysis. This means users can not only identify database discrepancies on a larger scale, but track changes overtime as it pertains to their personal research. This may assist in the identification of error patterns and will certainly make large scale sequence validation more advantageous; the implications of which can correspond with wherever research that utilizes sequencing is present.

Chapter II

Materials and Methods

The programmatic design, dependencies, flow, run-options, data utilizations and algorithms are outlined within this section. All source code and external utilities used to develop source code will be provided in the appendix. Software development practices were followed, though by nature implementations are constantly undergoing additions and refinements. The project can be followed live on github (https://github.com/ashbig/Annow).

Development Tools

Annow was developed using the Python programming language version 2.7 coded in the Enthought Canopy Integrated Development Environment (IDE). Coding and style conventions were followed according to the accepted PEP 8 Style Guide for Python Code (Brandl, 2016). Annow's Graphical User Interface (GUI) was developed in the same IDE using the open source module wxPython. NCBI's BLAST version 2.2.30 was utilized for the compilation of BLAST databases (module makeblastdb) as well as local alignments (module blastn, blastp). All associated libraries and imports used in Annow can be found in the appendix. PyInstaller version 3.2 was used to bundle required dependencies (blastn, blastp, makeblastdb, Python libraries, wxPython) into a single executable file for distribution, and source code is available on github.com at (https://github.com/ashbig/Annow).

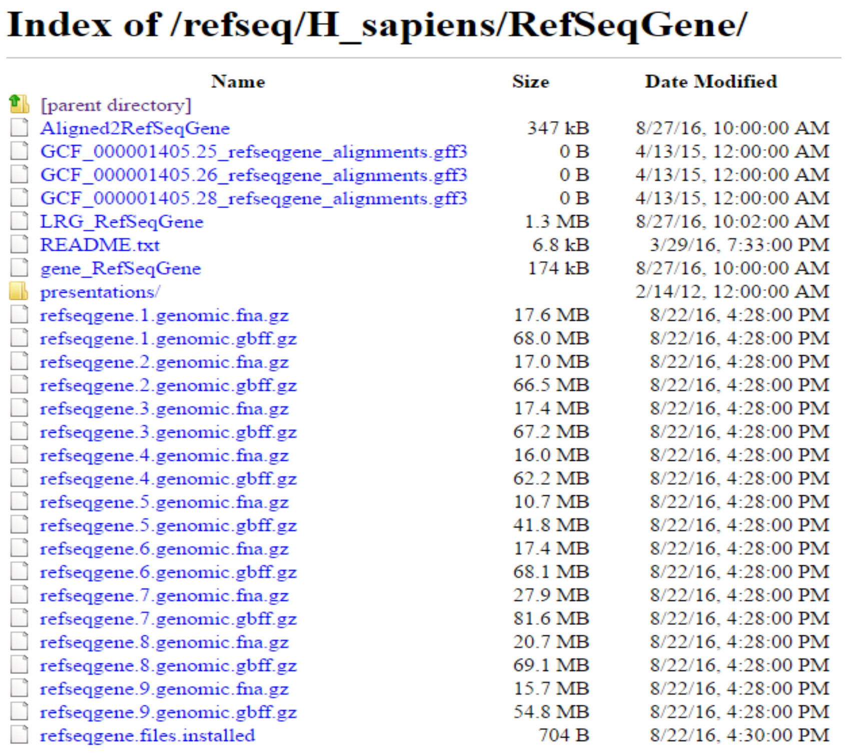
Design Considerations

Large scale sequence alignment and annotation modifications are most often done on high performance computing clusters (HPC) that allow for faster computational operations. This is because reading large files containing genetic sequences and performing analysis, such as sequence alignments, can be computationally taxing and therefore easiest to accomplish on systems typically much stronger than personal computers. It was the goal of this project to allow users to run Annow without any prior computational expertise. As such, it was designed on personal computer, and can be run on a personal computer with no specific requirements.

To achieve this goal programmatic functions used in Annow are conscious of the amount of computational memory consumed so as not to overload a system. Input and output files are read line by line, rather than reading entire files into computer memory, a resource that when unavailable will halt systems. This process was made possible by heavy use of Python 2.7 data structures, mainly dictionaries. Dictionaries provide a key and value system where new, or previous, annotations, or sequences can serve as the key or values. This enables only a small fraction of the total data (the annotation or sequence) to be housed in a computer’s memory as oppose to most available tools which will process the annotation and sequence relying on computational power over algorithmic implementation. Annow leverages this capability to retrieve relevant sequences only when needed, bringing gene annotation to the personal computer.

Programmatic Flow

Reference sequences and their corresponding annotations are made available by NCBI, Ensembl and other groups through what is referred to as a File Transfer Protocol site (FTP). Researchers can then access and download compressed files containing gene sequences. In most instances the sequences will be too large to be contained in a single file. As such, they split into several files containing a common file suffix (Figure 5). Each file is in the common format type FASTA, which uses the ">" character to distinguish sequence meta-data from the sequence itself.



Index of /refseq/H_sapiens/RefSeqGene/

Name	Size	Date Modified
[parent directory]		
Aligned2RefSeqGene	347 kB	8/27/16, 10:00:00 AM
GCF_000001405.25_refseqgene_alignments.gff3	0 B	4/13/15, 12:00:00 AM
GCF_000001405.26_refseqgene_alignments.gff3	0 B	4/13/15, 12:00:00 AM
GCF_000001405.28_refseqgene_alignments.gff3	0 B	4/13/15, 12:00:00 AM
LRG_RefSeqGene	1.3 MB	8/27/16, 10:02:00 AM
README.txt	6.8 kB	3/29/16, 7:33:00 PM
gene_RefSeqGene	174 kB	8/27/16, 10:00:00 AM
presentations/		2/14/12, 12:00:00 AM
refseqgene.1.genomic.fna.gz	17.6 MB	8/22/16, 4:28:00 PM
refseqgene.1.genomic.gbff.gz	68.0 MB	8/22/16, 4:28:00 PM
refseqgene.2.genomic.fna.gz	17.0 MB	8/22/16, 4:28:00 PM
refseqgene.2.genomic.gbff.gz	66.5 MB	8/22/16, 4:28:00 PM
refseqgene.3.genomic.fna.gz	17.4 MB	8/22/16, 4:28:00 PM
refseqgene.3.genomic.gbff.gz	67.2 MB	8/22/16, 4:28:00 PM
refseqgene.4.genomic.fna.gz	16.0 MB	8/22/16, 4:28:00 PM
refseqgene.4.genomic.gbff.gz	62.2 MB	8/22/16, 4:28:00 PM
refseqgene.5.genomic.fna.gz	10.7 MB	8/22/16, 4:28:00 PM
refseqgene.5.genomic.gbff.gz	41.8 MB	8/22/16, 4:28:00 PM
refseqgene.6.genomic.fna.gz	17.4 MB	8/22/16, 4:28:00 PM
refseqgene.6.genomic.gbff.gz	68.1 MB	8/22/16, 4:28:00 PM
refseqgene.7.genomic.fna.gz	27.9 MB	8/22/16, 4:28:00 PM
refseqgene.7.genomic.gbff.gz	81.6 MB	8/22/16, 4:28:00 PM
refseqgene.8.genomic.fna.gz	20.7 MB	8/22/16, 4:28:00 PM
refseqgene.8.genomic.gbff.gz	69.1 MB	8/22/16, 4:28:00 PM
refseqgene.9.genomic.fna.gz	15.7 MB	8/22/16, 4:28:00 PM
refseqgene.9.genomic.gbff.gz	54.8 MB	8/22/16, 4:28:00 PM
refseqgene.files.installed	704 B	8/22/16, 4:30:00 PM

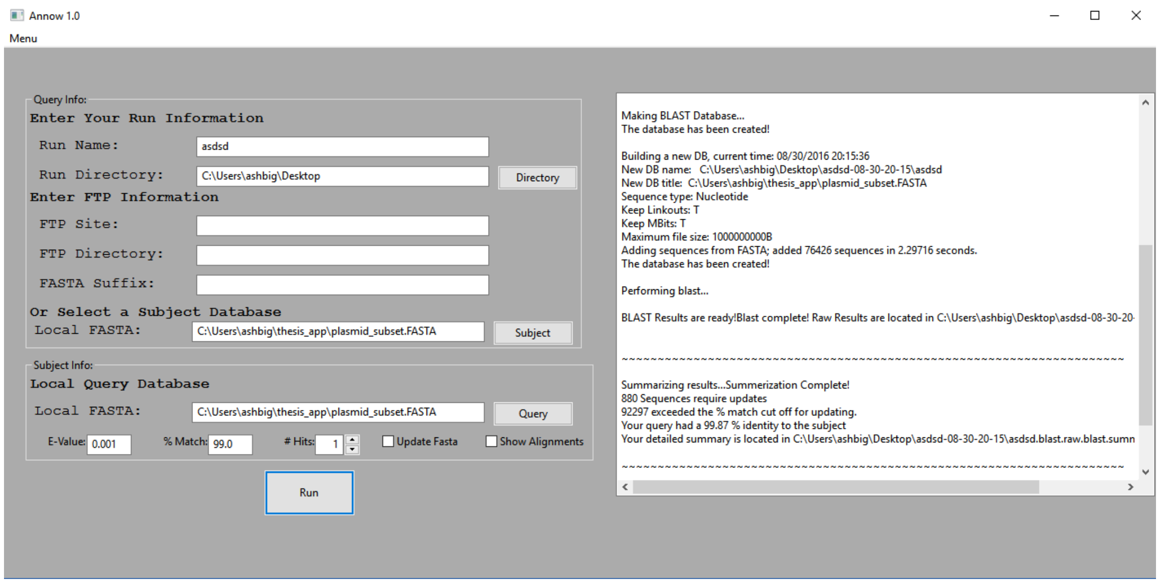
Figure 5. Example of common directory and file structure containing the latest RefSeq release. File suffixes remain consistent while prefixes denote the file.

Annow requires that the user provide the location to this FTP site, the directory on the ftp site in which the sequences are stored, as well as the file suffix (Figure 5). Using the Python library for FTP transfers Annow downloads all files in the given FTP directory to a user specified directory. Utilizing the Python library for file compression and decompression Annow proceeds to decompress all downloaded files and begins concatenating them into a single file. After concatenation is complete, all downloaded files are removed from the system to maintain only the required storage volume.

Should the user only wish to use Annow for its BLAST and annotation updating capabilities, they are given the option of providing a local FASTA format file.

Additionally, users are required to give the operation a name (Figure 6). This name will be used to segregate various runs of Annow and will be placed in a directory that's contains the operation name, date, hour and minute to allow users to track various operations.

Once the sequences that will be queried upon are ready, Annow performs a system call using Python's OS library to NCBI's makeblastdb module which generates BLAST databases from FASTA files specific to the method of analysis selected by the user (protein or nucleotide). Upon completion the user is provided with a summary, both printed in the GUI and maintained in a small local file residing in the user specified directory. The summary information includes what type of database was created, the name of the database, number of sequences in the database, as well as the total duration in seconds the BLAST database took to be built.



Query Info:
Enter Your Run Information
Run Name: asdsd
Run Directory: C:\Users\ashbig\Desktop
Enter FTP Information
FTP Site:
FTP Directory:
FASTA Suffix:
Or Select a Subject Database
Local FASTA: C:\Users\ashbig\thesis_app\plasmid_subset.FASTA

Subject Info:
Local Query Database
Local FASTA: C:\Users\ashbig\thesis_app\plasmid_subset.FASTA
E-Value: 0.001 % Match: 99.0 # Hits: 1 ☐ Update Fasta ☐ Show Alignments

```
Making BLAST Database...
The database has been created!

Building a new DB, current time: 08/30/2016 20:15:36
New DB name:   C:\Users\ashbig\Desktop\asdsd-08-30-20-15\asdsd
New DB title:  C:\Users\ashbig\thesis_app\plasmid_subset.FASTA
Sequence type: Nucleotide
Keep Linkouts: T
Keep MBits: T
Maximum file size: 1000000000B
Adding sequences from FASTA; added 76426 sequences in 2.29716 seconds.
The database has been created!

Performing blast...

BLAST Results are ready!Blast complete! Raw Results are located in C:\Users\ashbig\Desktop\asdsd-08-30-20-


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Summarizing results...Summerization Complete!
880 Sequences require updates
92297 exceeded the % match cut off for updating.
Your query had a 99.87 % identity to the subject
Your detailed summary is located in C:\Users\ashbig\Desktop\asdsd-08-30-20-15\asdsd.blast.raw.blast.sumn

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 6. Annow GUI. Instructions and specifications are contained within the appendix.

The user has several options for output which are parameter dependent. Annow will always output a summary of the annotations it has found that require updating. Additionally the user can chose to have a file containing alignments, which displays a base by base local alignment graphic, as well as generate an updated FASTA file containing only sequences which meet update criteria set by the user. Additional parameters and specifications can be found in the appendix and are further described below.

Runtime Options

Table 1. Annow Runtime Options

Parameter	Purpose
Subject	The set of sequences to serve as the database. Local or Remote Source
Query	The set of sequences to update annotations. Local source.
E-value	The rate to reduce randomness of observed sequences.
% Match	The cut-off value for which to display results.
# Hits	The number if possible matches to return for query sequences.
Protein	Switches the method of analyses from nucleotide to protein.
Short	Alters the program to accommodate sequences less than 30bp.
Alignments	Displays molecule by molecule alignment.

The various parameters a user can select and a brief explanation.

In Annow analyses are separated by a name input by the user. The name of the run is used to create a directory in the location designated by the user. The directory created is appended with month, day, hour and minute to ensure that analyses are kept separated (i.g, orf_protein-09-17-23-00). If either options is absent the program will halt and prompt the user to enter a name appropriately.

Query sequences can be either downloaded from a remote source, or input as a local source. For remote downloads the user is asked to provide the ftp site (i.g, ftp.ncbi.nlm.nih.gov/), the directory (i.g, /refseq/H_sapiens/mRNA_Prot/) and the file suffix (i.g, rna.fna.gz). For local sequences, either subject or query, the expectation is that the input will be in the FASTA format. Absent of all required query or a local sources the program will halt and prompt the user to enter a name appropriately.

Analyses can be evaluated with three parameters presented to the user. The first, e-value, is a measure of randomness. For example, an e-value of 0.001 states that there is a hundredth of a chance that the sequence shown is random. The second evaluation operator is a cut-off percentage. This value is used to remove unwanted results from the final output summary, as well as the updated FASTA and represents the overall percentage identity of a given sequence. For example, if the user requests a cut-off of 99%, only results greater than 99% alignment match will appear in the summary and the updated FASTA should the user request it. The last option is a value for the number of results per sequence to display, if the user selects a value of 5, up to 5 matches in a given database will be displayed.

Annow provides two additional methods of analysis beyond the default of nucleotide, as well as one additional method of viewing beyond a simple summary. This first method is protein analysis, which leverages NCBI's blastp program. If selected, Annow will translate the established subject and query from nucleotide sequences to amino acid sequences. This enables the user to determine if there is a possible change in the coding region itself. The second method of analysis is referred to as "short" and leverages NCBI's ability to alter its internal algorithm to blast short sequences, which is especially valuable for shRNA or crisper/cas9 analysis. The additional method of analysis viewing is an alignment file which displays the molecule by molecule alignment allowing the user to view alterations with more granularity.

PlasmID Datasets

To validate Annow's performance and future applications data from 243,151 plasmid sequences were retrieved from The Dana Farber/Harvard Cancer Center DNA Resource Core (DF/HCC) PlasmID Database (Zu, 2007). The DF/HCC DNA Resource Core is plasmid storage and distribution facility that intakes bacterial plasmids from various genome projects and publications. Their catalog has grown over time to provide several iterations of most protein coding human genes as well as their corresponding RNA interference (RNAi) constructs and currently houses data on over 378,406 constructs.

The plasmid sequences retrieved from the DF/HCC correspond to cDNA, short hairpin RNA (shRNA), and Human ORFeome constructs (ORF) and are limited to only those that are regard human genes and are currently available for distribution. The sequences provided with these constructs are derived at the time of publication or intake. Verification is done by the depositing group prior to intake and distribution by the DF/HCC and carried out by end read sequencing. Prior to distribution a plasmid leaving the DF/HCC will be sequenced once more and compared to the depositor provided sequence to ensure integrity. Should a construct fail this comparative sequencing analysis it is removed from circulation. This process does verify that the sequence matches depositor information. However, as this sequencing data is derived at intake, it may or may not accurately reflect the genes current day representation.

As constructs distributed by the DF/HCC undergo sequencing when arriving and prior to leaving the facility, it is believed that only a small amount of constructs relative to the entirety of the collection will require additional review. However, if only a small fraction of constructs across the collection can undergo annotation update or removal form circulation the non-profit organization can save time and labor hours and meet their global missions more effectively.

As the DF/HCC's data spans many years, releases, and a variety of annotations, it serves as an ideal test case. Overall programmatic performance can be assessed by viewing run times for alignments, database creation and various methods of analyses. Most sequencing, alignment, or even small lab projects will fall well below a plasmid distribution center's needs providing Annow with an excellent stress test. Further a successful demonstration of Annow can show justification for its application beyond just the tested datasets.

Chapter III

Results

DF/HCC data shows a majority of sequences maintain their annotation, but many should be reviewed or entirely removed. The datasets used provide validation of the given method be it nucleotide, protein or short sequence analyses and is described in detail below.

DF/HCC Annotation Curation

Constructs from various depositors were aggregated from the DF/HCC DNA Resource Core PlasmID database and analyzed for nucleotide similarity. Each deposit was analyzed so that it may be assessed against the correct reference distribution and concordance can then be returned to the DF/HCC to make an evaluation on construct distribution and curation.

cDNA Analysis

The DF/HCC currently has 57, 031 human cDNA constructs, of which 41,217 are available for distribution. The remaining 15, 604 are likely to have been removed after failed distribution attempt with errors discovered by either the DF/HCC or the receiver. To correlate the cDNA plasmid sequences to current annotations the latest CCDS sequence release was downloaded using Annow from the CCDS FTP server. The collection contains 31,394 sequences and the FASTA files were downloaded in just under 1 minute and 30 seconds and the BLAST database compiled in approximately 1.3 seconds. BLAST analysis with the DF/HCC dataset was completed with a summary in just over 50 seconds.

Of the 41,217 nucleotide analyzed sequences 38,988 showed a greater than 99 % match to a reference sequence in the CCDS. 814 matched at greater than 95%, 236 matched between 90-95%, 103 matched between 85-90%, 69 matched between 80-85%, 11 matched between 75-80%, 13 fell below 75% matches and 1,013 matched no records in the CCDS (Figure 7).

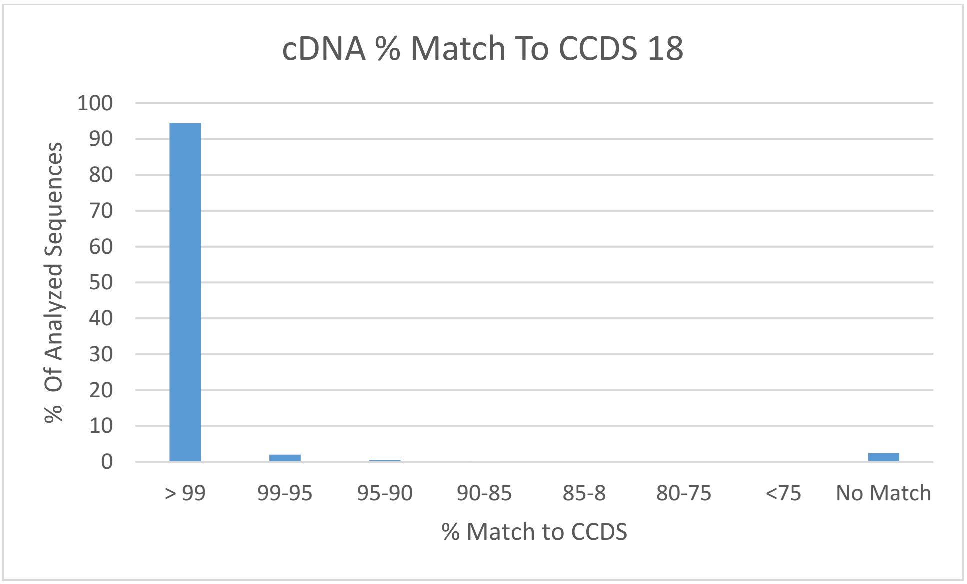


Figure 7. 41,217 cDNA sequences aligned to the latest CCDS release 18. Percentage of sequences aligned was high, while 5.5 % of the DF/HCC collection showed significant deviation.

The steady decrease in sequence affinity is logical considering some clones from this collection are over a decade in age. To assess if the majority of constructs producing no match reside in these older distributions a smaller subset from a single source, The Mammalian Gene Collection (MGC), which was deposited over a decade ago.

Over half of the DF/HCC cDNA constructs, 24,065 are MGC and represent some of the earliest DF/HCC distributions. The MGC's goal, a trans-National Institute of Health initiative, is to provide researchers with sequences validated full length cDNA's (MGC, 2004). The DF/HCC DNA Resource Core's goal is to distribute these constructs for as long as they remain accuracy. All 24, 065 MGC constructs were assessed by Annow for nucleotide concordance against the latest CCDS distribution (Figure 8). More than 80 % of these constructs aligned at greater than 99%, over 13% fewer than the total collection. Over 14 % MGC constructs showed no match to the current CCDS, an anticipated result due to the age of the collection.

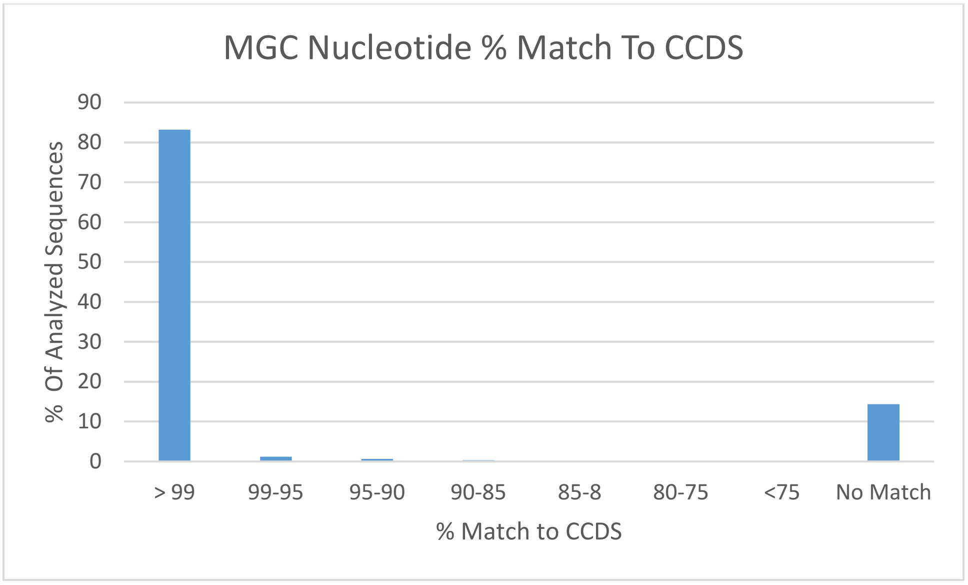


Figure 8. While 83.1 % of MGC constructs maintained their nucleotide integrity, less than 3 percent maintained alignment to the latest coding sequences.

Accounting for matches below 90% Annow has shown that approximately 6% of the available stock should be immediately sequestered from circulation as a result of the cDNA analysis alone. This prospective analysis will prevent the core from having to retrieve, validate and remove constructs manually. Further, the more constructs are handled the greater the opportunity for contamination rises via bench errors. Preemptively removing constructs that are no longer accurate gene representations reduces this risk and the stress of freezing and thawing constructs increasing their longevity.

ORF Analysis

The ORFeome Collaboration is a collection of many groups including the MGC, DF/HCC Resource Core, Wellcome Trust Sanger Institute, Dana Farber Cancer Institute-Center for Cancer Systems Biology and several others whose goal is to provide constructs that house the protein-coding region of genes only. Unlike the MGC collection, 5' and 3' untranslated regions (UTR) are purposefully omitted so that constructs are expression ready.

In their recent publication the ORFeome collaboration details their latest 8.1 release which houses 12,692 sequences covering 11,149 genes and the data set was retrieved from http://www.orfeomecollaboration.org (Wiemann, 2016). This release then serves as the best set to update annotations regarding DF/HCC's previous ORF deposits using codon analysis (blastp). The ORFeom collaboration has deposited a total of 49,791 Human ORF constructs, of which 5,701 have been removed from distribution. The remaining 44, 090 were correlated to the latest Human ORFeome release, 8.1 (Tools and Data, 2016).

Only 1.76% of all constructs analyzed in the DF/HCC ORF collection aligned at a percentage below 99.0. This includes the range from 98.99 to 0.0, with a total of 727 total constructs. 178 of this 727 showed an affinity greater than 95, while only 52 fellow below a 75% match to the current ORF release (Figure 9). 4,131 did fail to match entirely and as the DF/HCC maintains all collections form the ORF, this number likely reflects the number of discontinued sequences across all ORF distributions.

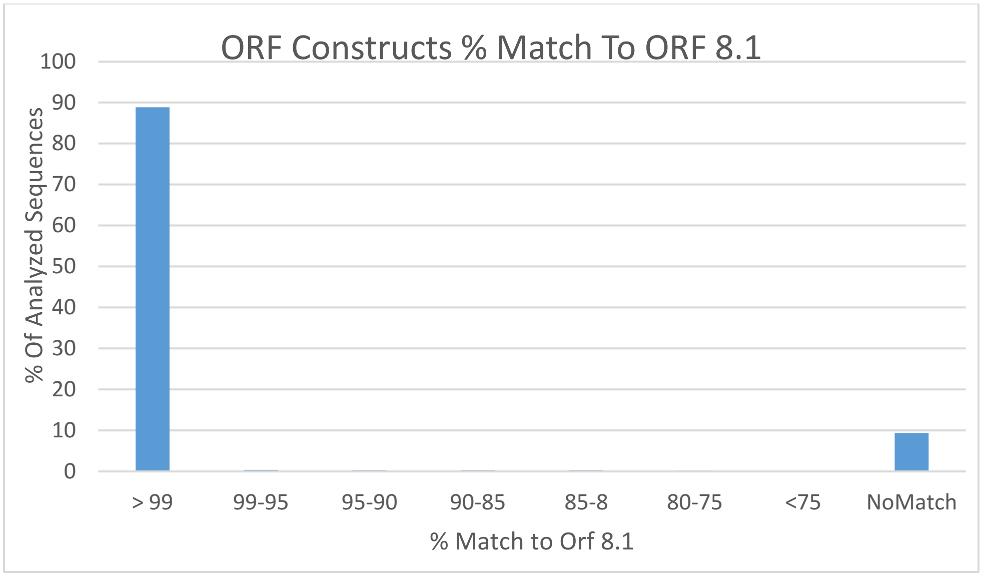


Figure 9. 44, 090 DF/HCC ORF constructs aligned to ORF 8.1. Almost all constructs analyzed showed significant matches at the protein level, or failed to match entirely with only 1.76% falling out of these two categories.

As query and subject sequences were translated prior to analyses the database compiled in an increased time of 3 minutes. The number of sequences analyzed by BLAST was the largest of the group and with translation of input sequences total analyses completed in 1 hour and 3 minutes.

shRNA Analysis

The DF/HCC distributes over 155, 000 shRNA constructs, 79,960 have human targets and 77,884 are mouse. shRNA, or RNAi, constructs are utilized heavily in the characterization of gene functions through post-translation silencing and by nature are significantly smaller than most cDNA's and ORF's averaging only 21 base pairs in length. This means the RNAi must have a high affinity to be effective and leaves little room for error during Sanger sequencing validation. Sequencing hairpin regions is notoriously difficult (Kieleczawa, 2005) and could be the contributing factor for such a large number of unavailable constructs (13,613). As such *in vivo* methods may provide the DF/HCC with a more reasonable method of annotation update.

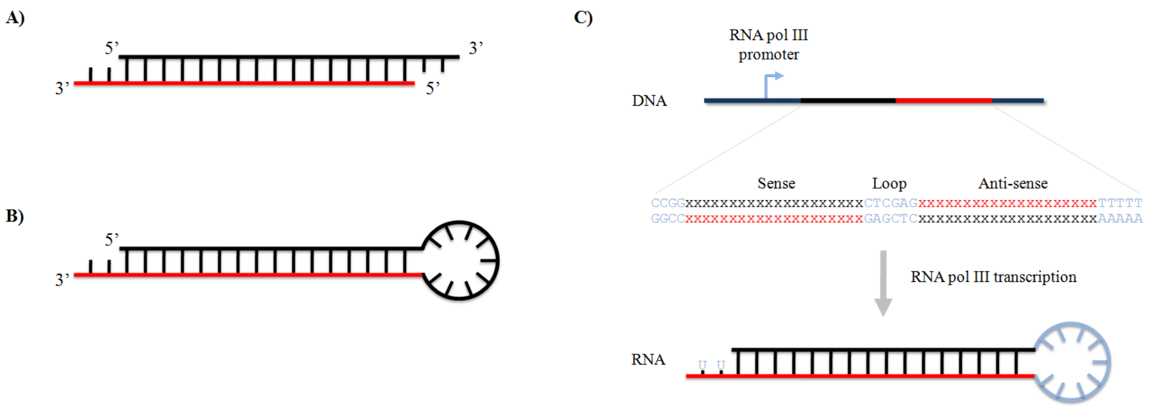

A)
5'
3'
3'
5'
B)
5'
3'
C)
RNA pol III promoter
DNA
Sense
Loop
Anti-sense
CCGGxxxxxxxxxxxxxxxxxxxxxCTCGAGxxxxxxxxxxxxxxxxxxxxxTTTTT
GGCCxxxxxxxxxxxxxxxxxxxxxGAGCTCxxxxxxxxxxxxxxxxxxxxxAAAAA
RNA pol III transcription
RNA
U U

Figure 10. Illustration of the hairpin structure seen in shRNA constructs. For sequence verification flanking sequences are removed, with only the loop (C) remaining.

To verify the distributed number of constructs, human and mouse shRNA's target regions (Figure 8, C) were aligned to the their respective NCBI RefSeq mRNA release. It was hoped that Annow would be able to effectively analyze these short fragments to determine sequence similarity percentage. However, only those sequences which matched the reference at 100% were able to be identified and at a greatly increased runtime. The extended run time is due to the algorithmic change in NCBI's alignment of short sequences, requiring each sequence to be aligned many times before an eventual exact match is selected. This increased run time may not be advantageous for analyzing the integrity of smaller data sets as a mechanism for shRNA biological reagent analysis exists in the form of UP-TORR (Yu, 2013).

Of the 79,960 human shRNA target regions analyzed 27, 303 showed perfect matches and maintained 100% affinity to their target. Of the 77, 884 mouse constructs analyzed 43,770 maintained their perfect target affinity over time. As this analyses only determines if the target is 100% accurate, 65.85 and 43.8 % of human and mouse constructs respectively should be further analyzed for data integrity to determine deposit accuracy and if the constructs still hold a high enough value of concordance to be effective.

While Annow cannot quickly and effectively assess non-perfect matches at this time, the observations derived from this shRNA analysis will inform future algorithmic development. As previously mentioned, Sanger analysis of hairpin constructs can require special protocols to separate the pinned structure making their verification and analyses more time consuming than longer constructs distributed by the core facility. A fast and accurate desktop method for technicians to quickly verify a targets validity post sequencing and prior to distribution would represent a cost and labor savings for the facility and with further refinement Annow could effectively meet this need.

Chapter IV

Discussion

It was Annow's aim to provide a mechanism to automate much of the process of in house gene annotation and make the alignment with consensus gene data easily attainable whether the research involves a thousand records or several hundred thousand. The results shown have verified not only the performance viability, but the ability of Annow to analyze and update sequences of nucleotides and proteins independent of sequence length. The application of which can be utilized with NCBI's, Ensembl's, or other resources available information and applied to data types such as amino acid, nucleotide or ribonucleotide sequences.

As shown by the DF/HCC's cDNA data set, Annow can handle the common project of nucleotide sequence alignment on a large scale at high speeds. It's easy to use and offline nature can provide researchers with a common blast utility on their personal computers. The cDNA analysis of both MGC and other cDNA clones demonstrates Annow's ability to unify sequence annotations regardless of origin. Aside from providing immediate gene verification with the latest annotation across a variety of sources, Annow can also be used to segregate legitimate sequencing results from faulty reads. The formatting of results in both the summary, and updated FASTA will save countless file manipulation operations that can be very quick for those with programming knowledge, but incredibly time consuming through manual manipulations.

The ORF dataset demonstrates Annow’s ability to analyze protein sequences effectively and even large dataset analyses can be accomplished in just over an hour allowing users more focused on protein interactions to easily visualize their sequences similarity at the amino acid level. This will also help organizations like the DF/HCC determine CDS percentage matches allowing the end user of their distribution network to shop their catalog more effectively. Further, Annow’s translation of nucleotide sequences can serve as a utility in its own right. Rather than copy and paste many sequences into a web source to convert nucleotide to amino acid sequences, a user could simply run Annow and the translated sequence file is provided to the user.

Annow’s ability to analyze short sequences was shown to be inadequate using shRNA constructs, but these observations have helped decide future development directions. Once implanted accurately the ability to analyze short sequences can be expanded to other applications such as primer alignment, crisper/cas9 and analysis of off target NGS results. If this implementation is achieved using simple string matching, rather than NCBI’s short-blast method it could dramatically reduce the runtime for these types of analyses.

The inclusion of sequence alignments in Annow was done so that in addition to annotation updating, the tool could also serve as an entirely offline version of BLAST. Easily distributed via a zip file and runnable by a simple double-click users have grown accustom to. This paired with the ability to easily segregate results in a straight forward manner allows for a wide range of users and it is the hope that many beyond the repository this tool was initially intended for will find Annow useful.

The development of Annow will contribute to the curation of over 170,000 plasmid sequences already in distribution by the DF/HCC. This will directly result in a cost and time resource saving in addition to providing their end users with a more reliable distribution center.

In all this project provides researchers around the globe with a free, downloadable program with which to compare local databases to a NCBI, Ensembl, or a reference database of their choice in a highly efficient manner. Users may find this tool most useful for curating large sequence databases yielding more accurate starting materials and references and minimizing costly hours spent doing sequence analysis. This tool will also prove useful for those with query heavy projects that run over a longer period of time than is possible on the web.

Perhaps the most exciting part of Annow will be its open source, heavily documented, easy to modify nature. While designed to be a tool for all users, by making all sources available online it is expected that the user community will find several more applications and modify source code as need. This methodology has help many programs blossom from single developer operations such as Annow, to large community driven projects that meet many needs beyond original intentions. For now Annow fills a very necessary gap in the informatics community, but with continued development has the potential to fill many more.

Appendix A

Annow User Manual

```
Annow version 1.0 Usage Manual

This software is intended to create NCBI BLAST nucleotide databases
from Local and Remote Sources for local querying.
This software can be used to update FASTA files with new
annotations, extract alignment metrics, and view alignments.
USAGE:

Run Name - Used for file and directory naming (i.e FASTA files,
results).  Spaces will be replaced by '_'.
Additionally, the current Month, Day, Hour, and Minute will be added
to your run name to further distinguish experiments.
example usage: "all_plamids_2013"


Run Directory - Select a local root directory to deposit results. A
new directory containing your run name and date will
be created within and output will be directed here.

prompted usage example: C:\Users\name\Desktop


FTP Site - If using a remote site enter the ftp url.

example usage: ftp.ncbi.nlm.nih.gov


FTP Directory - If using a remote site, enter the directory which
the files you wish to download are located.

example usage: /refseq/H_sapiens/RefSeqGene/


FTP Suffix - If using a remote site a file suffix is required. The
extension of this file should .gz and all
files ending in the provided suffix will be downloaded, be specific.

example usage for a single file: refseqgene1.genomic.fna.gz
example usage for multiple files: .genomic.fna.gz
example usage for all files: .gz


Local FASTA (subject) - You may use a local .FASTA rather than
remote. The file must be conventionally formatted
to build correctly and will serve as your BLAST database (subject).
```

prompted usage example: C:\Users\name\Desktop\mysubject.fasta

Local FASTA (query) - A fasta file that you would like to compare is required and must be conventionally formatted.

prompted usage example: C:\Users\name\Desktop\myquery.fasta

E-Value - This describes the number of hits one can expect to see by chance. Any decimal value can be entered,
the higher values increase runtime.

default value = 0.001
usage: 0.0002

% Match - This must be a % value and will be used in conjunction with the following Update FASTA parameter.
Any values BELOW this threshold will have annotation updated.

default value = 99.00
usage : 75.00

# Hits - This must be an whole number and will determine how many results to return while running BLAST.
A greater number of hits requested increases run time.

default value = 1
usage = 2

Protein - When checked will translated subject and query sequences prior to run, analyzing amino acid
sequences rather than nucleotide.

Short - Will run blast on short sequences (<30 bp).

Update FASTA - When this box is checked a FASTA file containing only updated sequences will be generated
in the results directory. They will be annotated by > OLD ANNOTATION | NEW ANNOTATION

default = unchecked

Alignments - This process will run blast again, this time generating a text file with local sequence alignments.
Selecting this will increase run by 2 fold.

default = unchecked

RUNNING Annow:

When all parameters are entered correctly press RUN to begin. This process is done concurrently
and the UI may become unresponsive during data processing. However any errors will be reported in screen,
and absent of this you may assume the process is ongoing. Depending on usage operations may take several hours or longer.

Version 1.0 is a stable release and new version can be found at https://github.com/ashbig/Annow/

Append B

Graphical User Interface

```
# Created by Ashkan Bigdeli 2016
# Annow.py
# Main UI and calls for data processing

# wx is external UI module
import wx, os
from src import pid_etr
from time import strftime


class ManualWindow(wx.Frame):
      #for manual page display
      def __init__(self):
           """Constructor"""
           wx.Frame.__init__(self, None, title="Annow User Manual", size=(720, 500))
           panel = wx.Panel(self)
           txt =wx.TextCtrl(panel, size=(700,450), style=wx.TE_MULTILINE | wx.TE_READONLY |wx.EXPAND)
           man = pid_etr.resource_path("README.TXT")
           with open(man) as readme:
             for line in readme:
                txt.AppendText(line)

class MainWindow(wx.Frame):

    def __init__(self,parent, title):
        wx.Frame.__init__(self,parent,title = title, size = (1300,650))


        panel = wx.Panel(self, -1)
        panel.SetBackgroundColour('#ABABAB')

        #set up standard menu options
        filemenu= wx.Menu()
        menuManual= filemenu.Append(wx.NewId(), "Manual"," Usage Guide")
        menuAbout= filemenu.Append(wx.ID_ABOUT, "&About"," Information about this program")
        menuExit = filemenu.Append(wx.ID_EXIT,"E&xit"," Terminate the program")

        #creating and binding the menu
        menuBar = wx.MenuBar()
        menuBar.Append(filemenu,"&Menu") # Adding the "filemenu" to the MenuBar
        self.SetMenuBar(menuBar)

        self.Bind(wx.EVT_MENU, self.OnManual, menuManual)
        self.Bind(wx.EVT_MENU, self.OnExit, menuExit)
        self.Bind(wx.EVT_MENU, self.OnAbout, menuAbout)

        #fonts
        entry_font = wx.Font(12, wx.MODERN, wx.NORMAL, wx.NORMAL)
        header_font = wx.Font(12,wx.MODERN, wx.NORMAL, wx.BOLD)


        #set up UI for all run information including event binding, sizing
        run_info = wx.StaticText(panel, label = "Enter Your Run Information")
        run_info.SetFont(header_font)

        #run name layout and bindings
        lblrun = wx.StaticText(panel, label = "Run Name: ", size=(175, -1))
        lblrun.SetFont(entry_font)
        self.editrun = wx.TextCtrl(panel, value="", size=(326, -1))
        ri_sizer = wx.BoxSizer(wx.HORIZONTAL)
        ri_sizer.AddSpacer(10)
        ri_sizer.Add(lblrun)
        ri_sizer.Add(self.editrun)

        #run directory layout and bindings
        lblrun_dir = wx.StaticText(panel, label = "Run Directory: ", size=(175, -1))
        lblrun_dir.SetFont(entry_font)
        self.editrun_dir = wx.TextCtrl(panel, value = "", size=(326, -1))
        self.dir_button = wx.Button(panel, label="Directory")
        self.dir_button.Bind(wx.EVT_BUTTON, self.opendir)
        rd_sizer=wx.BoxSizer(wx.HORIZONTAL)
        rd_sizer.AddSpacer(10)
        rd_sizer.Add(lblrun_dir)
        rd_sizer.Add(self.editrun_dir)
        rd_sizer.AddSpacer(10)
        rd_sizer.Add(self.dir_button)

        #ftp layout and bindings
        ftp_info = wx.StaticText(panel, label = "Enter FTP Information")
        ftp_info.SetFont(header_font)

        self.ftp_site = wx.StaticText(panel, label = "FTP Site: ", size=(175, -1))
        self.ftp_site.SetFont(entry_font)
        self.edit_ftp = wx.TextCtrl(panel, value = "", size=(326, -1))
        ftp_sizer=wx.BoxSizer(wx.HORIZONTAL)
        ftp_sizer.AddSpacer(10)
        ftp_sizer.Add(self.ftp_site)
        ftp_sizer.Add(self.edit_ftp)

        ftp_dir = wx.StaticText(panel, label = "FTP Directory: ", size=(175, -1))
        ftp_dir.SetFont(entry_font)
        self.edit_ftp_dir = wx.TextCtrl(panel, value = "", size=(326, -1))
        ftp_dir_sizer=wx.BoxSizer(wx.HORIZONTAL)
        ftp_dir_sizer.AddSpacer(10)
        ftp_dir_sizer.Add(ftp_dir)
        ftp_dir_sizer.Add(self.edit_ftp_dir)

        ftp_suffix = wx.StaticText(panel, label = "FASTA Suffix: ", size=(175, -1))
        ftp_suffix.SetFont(entry_font)
        self.edit_suffix = wx.TextCtrl(panel, value = "", size=(326, -1))
               ftp_suffix_sizer=wx.BoxSizer(wx.HORIZONTAL)
               ftp_suffix_sizer.AddSpacer(10)
```

```python
        ftp_suffix_sizer.Add(ftp_suffix)
        ftp_suffix_sizer.Add(self.edit_suffix)


        #local fasta layout and bindings
        fasta_info = wx.StaticText(panel, label = "Or Select a Subject Database")
        fasta_info.SetFont(header_font)
        local_fasta = wx.StaticText(panel, label = "Local FASTA: ", size=(175, -1))
        local_fasta.SetFont(entry_font)
        self.edit_fasta = wx.TextCtrl(panel, value = "", size=(326, -1))
        self.subject_button = wx.Button(panel, label="Subject")
        self.subject_button.Bind(wx.EVT_BUTTON, self.openfile)


        local_fasta_sizer=wx.BoxSizer(wx.HORIZONTAL)
        local_fasta_sizer.AddSpacer(5)
        local_fasta_sizer.Add(local_fasta)
        local_fasta_sizer.Add(self.edit_fasta)
        local_fasta_sizer.AddSpacer(10)
        local_fasta_sizer.Add(self.subject_button)


        # query fasta layout and bindings
        query_info = wx.StaticText(panel, label = "Local Query Database")
        query_info.SetFont(header_font)
        query = wx.StaticText(panel, label = "Local FASTA: ", size=(175, -1))
        query.SetFont(entry_font)
        self.edit_query = wx.TextCtrl(panel, value = "", size=(326, -1))
        self.query_button = wx.Button(panel, label="Query")
        self.query_button.Bind(wx.EVT_BUTTON, self.openfile)
        query_sizer=wx.BoxSizer(wx.HORIZONTAL)
        query_sizer.AddSpacer(5)
        query_sizer.Add(query)
        query_sizer.Add(self.edit_query)
        query_sizer.AddSpacer(10)
        query_sizer.Add(self.query_button)
        query_sizer.AddSpacer(5)



        #add all run, ftp, local/remote, query to a single sizer for UI
        info_sizer=wx.BoxSizer(wx.VERTICAL)
        info_sizer.Add(run_info)
        info_sizer.AddSpacer(10)
        info_sizer.Add(ri_sizer)
        info_sizer.AddSpacer(10)
        info_sizer.Add(rd_sizer)
        info_sizer.Add(ftp_info)
        info_sizer.AddSpacer(10)
        info_sizer.Add(ftp_sizer)
        info_sizer.AddSpacer(10)
        info_sizer.Add(ftp_dir_sizer)
        info_sizer.AddSpacer(10)
```

```
                   info_sizer.Add(ftp_suffix_sizer)
                   info_sizer.AddSpacer(10)
                   info_sizer.Add(fasta_info)
                   info_sizer.Add(local_fasta_sizer)

                   sbox = wx.StaticBox(panel, -1, 'Query Info:')
                   sboxSizer = wx.StaticBoxSizer(sbox, wx.VERTICAL)
                   sboxSizer.Add(info_sizer,0,wx.EXPAND, 200)


                   # set up run options including sizing, event bindings
                   eval_in = wx.StaticText(panel,label ='E-Value:')
                   self.edit_eval= wx.TextCtrl(panel,value="0.001",size=(45,-1))
                   eval_box = wx.BoxSizer(wx.HORIZONTAL)
                   eval_box.AddSpacer(5)
                   eval_box.Add(eval_in)
                   eval_box.Add(self.edit_eval)

                   self.num_hits = wx.StaticText(panel,label ='# Hits:')
                   self.hits = wx.SpinCtrl(panel, 1, min=1, max = 5,size=(45,-1))
                   self.hits.SetValue(1)
                   hits_box = wx.BoxSizer(wx.HORIZONTAL)
                   hits_box.AddSpacer(5)
                   hits_box.Add(self.num_hits)
                   hits_box.Add(self.hits)

                   self.percent_match = wx.StaticText(panel,label ='% Match:')
                   self.match= wx.TextCtrl(panel,value="0.001",size=(45,-1))
                   self.match.SetValue("99.0")
                   match_box = wx.BoxSizer(wx.HORIZONTAL)
                   match_box.AddSpacer(5)
                   match_box.Add(self.percent_match)
                   match_box.Add(self.match)

                   self.update = wx.CheckBox(panel, -1, 'Update FASTA')
                   self.show_align= wx.CheckBox(panel,-1, 'Alignments')
                   self.show_align.Bind(wx.EVT_CHECKBOX, self.alignment_warning)


                   self.pblast = wx.CheckBox(panel, -1, 'Protein')
                   self.short = wx.CheckBox(panel, -1, 'Short')


                   # add run options to a sizer for layout managment

                   options = wx.BoxSizer(wx.HORIZONTAL)
                   options.AddSpacer(2)
                   options.Add(eval_box)
                   options.AddSpacer(11)
                   options.Add(match_box)
                   options.AddSpacer(11)
                   options.Add(hits_box)
                   options.AddSpacer(11)
                   options.Add(self.pblast)
                   options.AddSpacer(11)
                   options.Add(self.short)
                   options.AddSpacer(11)
                   options.Add(self.update)
                   options.AddSpacer(10)
                   options.Add(self.show_align)

```

```python
        #aggreagate query info and options into one sizer for layout managment
        sbox2 = wx.StaticBox(panel, -1, 'Subject Info:')
        sboxSizer2 = wx.StaticBoxSizer(sbox2, wx.VERTICAL)
        sboxSizer2.Add(query_info)
        sboxSizer2.AddSpacer(10)
        sboxSizer2.Add(query_sizer)
        sboxSizer2.AddSpacer(20)


        sbox3 = wx.StaticBox(panel, -1, 'Options:')
        sboxSizer3 = wx.StaticBoxSizer(sbox3, wx.HORIZONTAL)
        sboxSizer3.Add(options)
        #aggregate all run panels and results panel and options into single panel

        run_sizer= wx.BoxSizer(wx.VERTICAL)
        run_sizer.Add(sboxSizer, wx.ALIGN_CENTER)
        run_sizer.AddSpacer(10)
        run_sizer.Add(sboxSizer2)
        run_sizer.AddSpacer(10)
        run_sizer.Add(sboxSizer3)
        run_sizer.AddSpacer(20)
        self.run_button = wx.Button(panel, label="Run",size=(100,50))
        self.run_button.Bind(wx.EVT_BUTTON,self.run)
        run_sizer.Add(self.run_button,0,wx.ALIGN_CENTRE)
        self.result_box = wx.TextCtrl(panel, size=(600,500), style=wx.TE_MULTILINE | wx.TE_READONLY |wx.HSCROLL)


        #set layout for run and result panels
        main_sizer=wx.BoxSizer(wx.HORIZONTAL)
        main_sizer.AddSpacer(25)
        main_sizer.Add(run_sizer,0,wx.ALIGN_LEFT,wx.EXPAND)
        main_sizer.AddSpacer(25)
        main_sizer.Add(self.result_box,0,wx.ALIGN_RIGHT,wx.EXPAND)


        layout_sizer=wx.BoxSizer(wx.VERTICAL)
        layout_sizer.AddSpacer(50)
        layout_sizer.Add(main_sizer,1,wx.EXPAND)
        layout_sizer.AddSpacer(50)


        #set layout and display UI
        self.SetSizer(layout_sizer)
        self.Centre()
        self.Show()

    # Menu Item events
    def OnAbout(self,e):
        # Create a message dialog box
        dlg = wx.MessageDialog(self, "Annow version 1.0 Gene Annotation Software\nCreated and maintained by Ashkan Bigdeli\nFree to use and distribute\nSource code and the latest version can be found on Github\nhttps://github.com/ashbig/Annow/", "Annow", wx.OK)
        dlg.ShowModal() # Shows it
        dlg.Destroy() # finally destroy it when finished.

    def OnManual(self,e):
        # Create a message dialog box
        man_page = ManualWindow()
        man_page.Show()


    def OnExit(self,e):
        self.Close(True)  # Close the frame.

    # GUI Events
    def openfile(self, event):
       dlg = wx.FileDialog(self, "Choose a file", os.getcwd(), "", "*.*", wx.OPEN)
       label = event.GetEventObject().GetLabel()
       if dlg.ShowModal() == wx.ID_OK:
                path = dlg.GetPath()
                if label == "Subject":
                    self.edit_fasta.SetValue(path)
                if label == "Query":
                    self.edit_query.SetValue(path)
       dlg.Destroy()

    def opendir(self, event):
        dlg = wx.DirDialog(self, "Choose a directory:", style=wx.DD_DEFAULT_STYLE | wx.DD_NEW_DIR_BUTTON)
        label = event.GetEventObject().GetLabel()
        if dlg.ShowModal() == wx.ID_OK:
            if label == "Directory":
                self.editrun_dir.SetValue(dlg.GetPath())
        dlg.Destroy()


    def alignment_warning(self,e):
        if self.show_align.IsChecked():
            dlg = wx.MessageDialog(self, 'Generating Alignments Will Increase Run Time 2x!', 'Annow 1.0', wx.OK|wx.ICON_INFORMATION)
            dlg.ShowModal()
            dlg.Destroy()

    # Data processing event
    def run(self, event):
        run_name = self.editrun.GetValue()
        run_name = run_name.replace(" ", "_")
        run_dir = self.editrun_dir.GetValue()
        ftp_url = self.edit_ftp.GetValue()
        ftp_dir = self.edit_ftp_dir.GetValue()
        ftp_suffix = self.edit_suffix.GetValue()
        local_subject = self.edit_fasta.GetValue()
        query_db = self.edit_query.GetValue()
        evalue = self.edit_eval.GetValue()
        hit_val = self.hits.GetValue()
        alignments = self.show_align.GetValue()
        update_fasta = self.update.GetValue()
        match_val = self.match.GetValue()
        protein = self.pblast.GetValue()
        short_blast = self.short.GetValue()

        #store file name for updating fasta if protein alignments are done
        query_orig = self.edit_query.GetValue()

        #check to see if we have input from the user
        if len(run_name) < 1:
```

```python
            self.editrun.SetValue("You Must Enter A Run Name!")
            return
        if len(run_dir) < 1:
            self.editrun_dir.SetValue("You Must Enter An Output Directory!")
            return
        if len(ftp_url) <1 or len(ftp_dir) <1 or len(ftp_suffix) < 1:
            if len(local_subject) < 1:
                self.edit_ftp.SetValue("You Must Enter Remote or Subject Database Completely!")
                return
        if len(query_db) < 1:
            self.edit_query.SetValue("You Must Enter A Query FASTA!")
            return


        #display run params for user
        self.result_box.AppendText("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
        self.result_box.AppendText("\n\nRUN PARAMETERS:\n")
        self.result_box.AppendText("Run Name: " + run_name +"\n")
        self.result_box.AppendText("Run Directory: " + run_dir +"\n")
        self.result_box.AppendText("FTP Site: " + ftp_url + "\n")
        self.result_box.AppendText("FTP Directory: " + ftp_dir + '\n')
        self.result_box.AppendText("FTP File Suffix: " + ftp_suffix + '\n')
        self.result_box.AppendText("Local Subject DB: " + local_subject + '\n')
        self.result_box.AppendText("Query Database: " + query_db + '\n')
        self.result_box.AppendText("Blast E-Value: " + evalue + '\n')
        self.result_box.AppendText("Match Criteria for Updating: " + str(match_val) + '\n')
        self.result_box.AppendText("Sequence Hits: " + str(hit_val) + '\n')
        self.result_box.AppendText("Generate Alignments: " + str(alignments) + '\n')
        self.result_box.AppendText("Update Input Query: " + str(update_fasta) + '\n\n')
        self.result_box.AppendText("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

        # make run specific directory
        time = strftime("-%m-%d-%H-%M")
        new_run_path = os.path.join(run_dir,(run_name + time))
        os.mkdir(new_run_path, 0755)

        #if ftp, download, unzip, concatenate and generate a file fasta file
        if len(local_subject) < 1:

            self.result_box.AppendText("Downloading FASTA files...\n")
            dwnld_msg = pid_etr.download(ftp_suffix, ftp_url, ftp_dir, new_run_path)
            self.result_box.AppendText(dwnld_msg + "\n")
            if "error" in dwnld_msg:
                return

            self.result_box.AppendText("Unzipping Downloaded FASTA's...\n")
            print new_run_path
```

```
                    unzip_msg = pid_etr.unzip(new_run_path)
                    self.result_box.AppendText(unzip_msg + "\n")
                    if "error" in unzip_msg:
                        return

                    self.result_box.AppendText("Removing Intermmediate Files...\n")
                    remove_msg = pid_etr.remove(ftp_suffix, new_run_path)
                    self.result_box.AppendText(remove_msg + "\n")
                    if "error" in remove_msg:
                        return

                    self.result_box.AppendText("Concatinating Files...\n")
                    concat = pid_etr.concat_fasta(run_name, new_run_path)
                    concat_msg = concat[0]
                    self.result_box.AppendText(concat_msg + "\n")
                    if "error" in concat_msg:
                        return
                    local_subject= concat[1]

                    self.result_box.AppendText("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

                blast_task = ""
                if short_blast:
                    blast_task = "-task blastn-short"
                db_type = "nucl"
                blast_type = "blastn"
                if protein:
                    db_type = "prot"
                    blast_type="blastp"
                    if short_blast:
                        blast_task = "-task blastp-short"
                    run_name = run_name + '.' + db_type
                    self.result_box.AppendText("\n\nTranslating your subject...\n")
                    subject_translate = pid_etr.translate(local_subject,run_dir)

                    translate_msg = subject_translate[0]
                    self.result_box.AppendText(translate_msg)
                    if "error" in translate_msg:
                        return
                    local_subject = subject_translate[1]

                    self.result_box.AppendText("\n\nTranslating your query subject...\n")
                    query_translate = pid_etr.translate(query_db, run_dir)
                    translate_msg = query_translate[0]
                    self.result_box.AppendText(translate_msg)
                    if "error" in translate_msg:
                        return
                    query_db = query_translate[1]




                #create blast database and update user
                self.result_box.AppendText("\n\nMaking BLAST Database...\n")
                make_blast_db = pid_etr.create_db(new_run_path, local_subject, run_name, db_type)
                blast_db_msg = make_blast_db[0]
```

```
                self.result_box.AppendText(blast_db_msg)
                if "error" in blast_db_msg:
                    return

                blast_db = make_blast_db[1]
                blast_db_summary = make_blast_db[2]
                with open(blast_db_summary) as summary:
                    for line in summary:
                        self.result_box.AppendText(line)

                self.result_box.AppendText(blast_db_msg)

                # perform blast and update user
                self.result_box.AppendText("\n\nPerforming blast...\n\n")
                results_tuple = pid_etr.perform_blast(query_db, blast_db, new_run_path, run_name, evalue, str(hit_val), blast_type, blast_task)
                result_msg = results_tuple[0]
                self.result_box.AppendText(result_msg)
                if "error" in result_msg:
                    return
                results = results_tuple[1]

                if os.path.isfile(results) is False:
                     self.result_box.AppendText("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n\n\n")
                     self.result_box.AppendText("\n\t\tBlast did not generate a file, there may be no matches, or an issue with the input FASTA!\n\n\n\n")
                     return
                self.result_box.AppendText("Blast complete! Raw Results are located in " + results + "\n\n")

                self.result_box.AppendText("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

                # summarize results and update user
                self.result_box.AppendText("\nSummarizing results...")
                summarize_tuple = pid_etr.summarize_results(results, match_val,str(hit_val))
                summarize_msg = summarize_tuple[0]
                self.result_box.AppendText(summarize_msg)
                if "error" in summarize_msg:
                    return
                summary = summarize_tuple[1]
                new_annotation = summarize_tuple[2]
                if "Concordance" in new_annotation:
                    update_fasta=False
                    self.result_box.AppendText("\n" + new_annotation + "\n")

                # update fasta is asked
                if update_fasta:

                    self.result_box.AppendText("\n\n\nGenerating a new FASTA with updated annotations...\n\n")
                    update_tuple = pid_etr.update_fasta(run_dir, run_name, new_annotation, query_orig)
                    update_message = update_tuple[0]
                    self.result_box.AppendText(update_message)
                    if "error" in update_message:
                        return

```

```
                    self.result_box.AppendText("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

                # provide alignments if asked
                if alignments:
                    self.result_box.AppendText("\nGenerating an alignments file...")
                    alignments_tuple = pid_etr.perform_blast_align(query_db, blast_db, new_run_path, run_name, evalue, str(hit_val), blast_type, blast_task)
                    align_message = alignments_tuple[0]
                    self.result_box.AppendText(align_message)
                    if "error" in align_message:
                        return

                self.result_box.AppendText("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
                self.result_box.AppendText("\n\nYour run is complete! Check " + run_dir + " for your results!\n\n\n")



        #run app
        app = wx.App(False)
        frame = MainWindow(None, "Annow 1.0")
        app.MainLoop()
```

Appendix C

Data Processing

```python
# Created By Ashkan Bigdeli 2016
# pid_etr.py
#
# A repository housing all required methods for updating sequences id's
# with new reference annotations

import os, gzip, glob, traceback,sys, textwrap
from ftplib import FTP
from collections import OrderedDict



# resource path
#
# @param1 = path of operations
#
#  returns a path for blast executables when running as an application
def resource_path(relative_path):
    """ Get absolute path to resource, works for dev and for PyInstaller """
    if hasattr(sys, '_MEIPASS'):
        return os.path.join(sys._MEIPASS, relative_path)

    return os.path.join(os.path.abspath("."), relative_path)

#provide makeblastdb path
makeblastdb = resource_path("makeblastdb")


# download
#
# @param1 = file_id: suffix of files ot be downloaded.
# @param2 = ftp_url: ftp from which to download
# @param3 = ftp_dir: directory from ftp to download
# @param4 = run_name: for naming scheme
# @param5 = run_dir: the directory to download to
#
# return = a message that the operation is complete
#
# Will download all specified files to a designated folder.
def download(file_id,ftp_url, ftp_dir, run_dir):

    try:
        ftp = FTP(ftp_url)
        ftp.login()
        # Change directory in ftp to navigate to desired genome
        ftp.cwd(ftp_dir)
        filenames = ftp.nlst()

        for filename in filenames:
            if filename.endswith(file_id):
                local_filename = os.path.join(run_dir, filename)
                file = open(local_filename, 'wb')
                ftp.retrbinary('RETR ' + filename, file.write)
                file.close()
        return "Downloading Complete!"
    except:
        return "Download Failed! Check the error below.\n" + traceback.format_exc()


# unzip
#
# @param1 = the directory to find files for decompression
#
# return = a message that the operation is complete
#
# Method will unzip all compressed files in given directory
def unzip(run_dir):
    try:
        # for each file in the directory
        for src_name in glob.glob(os.path.join(run_dir, '*.gz')):
            base = os.path.basename(src_name)
            dest_name = os.path.join(run_dir, base[:-3])
            with gzip.open(src_name, 'rb') as infile:
                with open(dest_name, 'wb') as outfile:
                  for line in infile:
                      outfile.write(line)
                os.chmod(dest_name, 0775)
        return "Unzip Complete!"
    except:
        return "The downloaded files could not be decompressed! Check the error blow. \n" + traceback.format_exc()

# remove
#
# @param1 = suffix: file type to be removed.
# @param2 = download_to: folder to remove downloaded files.
#
# return = a message the operation is complete
#
# Removes all given file types from folder.
def remove(suffix, run_dir):
    try:
        files_in_path = run_dir + "/*" + suffix
        files = glob.glob(files_in_path)
        for f in files:
            os.remove(f)
        return "Intermmediate Files Removed!"
    except :
        return "We failed to remove intermmediate files! Check the error below.\n" + traceback.format_exc()

       # concat fasta
       #
       # @param1 = filenames: all files to be concatenated.
       # @param2 = download_to: folder to concatenate files.
       #
       # return = a message the operation is complete
       #
       # Opens ALL files in folder and writes them to one .fasta file.
       def concat_fasta(run_name, run_dir):
           # create file list of all files to concatenate
           files_in_path = run_dir + '/*'
           filenames = glob.glob(files_in_path)
           fasta_path = os.path.join(run_dir,(run_name + '.fasta'))
           try:
               with open(fasta_path, 'w') as outfile:
                   for filename in filenames:
                       with open(filename) as infile:
                           for line in infile:
                               outfile.write(line)

                       infile.close()
                       os.remove(filename)
               outfile.close()
               return ("Concatentation Complete! Intermediate Files Removed!", fasta_path)
           except:
               return ("FASTA files could not be concatenated! Check the error blow. \n" + traceback.format_exc(), "", "")

       # create_db
       #
       # @param1 = run_dir:  the directory of operation
       # @param2 = fasta: the file for database creation
       # @param3 = run_name: for file naming the operation
       #
       # return = tuple = message, summary of database compilation, location of blast database
       #
       # Makes an external call to local blast program and creates desired blast database.
       def create_db(run_dir, fasta, run_name, db_type):
           try:
               db_summary = os.path.join(run_dir, (run_name + '.db_summary.txt'))
               db_location = os.path.join(run_dir, run_name)
               os.popen(makeblastdb + ' -in ' + fasta + ' -dbtype ' + db_type + ' -out ' + db_location+ ' > ' + db_summary)
               files_tuple = ("The database has been created!", db_location, db_summary)
               return files_tuple
           except:
               return ("The Database could not be created! Check the error below. \n" + traceback.format_exec(), "", "")

       # perform_blast
       #
       # @param1 = query_db: the fasta used for comparison
       # @param2 = blast_db: the blast database
       # @param3 = run_dir: the directory of operation
       # @param4 = run_name: for file naming operation
       # @param5 = evalue: the likely hood of randomness value
       # @param6 = hits: the number of hits returned per sequence
       # @param7 = blast_type: the type of blast to perform
       # @param8 = task: short blast if requested
       #
       # return = tuple = an operation message, a tab delimited file of hits in the subject_db
       #
       # Performs NCBI local alignment blast and returns a bare tab delimited results file
       def perform_blast(query_db, blast_db, run_dir, run_name, evalue, hits, blast_type, task):
           try:
               if blast_type == 'blastp':
                   blast_type = resource_path("blastp.exe")
               if blast_type == 'blastn':
                   blast_type = resource_path("blastn.exe")

               results = os.path.join(run_dir, (run_name + '.blast.raw.txt'))

               os.popen( blast_type + ' -db ' + blast_db + ' -query ' + query_db + ' -out ' + results +
                     ' -max_target_seqs ' + hits + ' -max_hsps ' + hits + ' -evalue ' + evalue +' ' + task +
                     ' -outfmt "6 qgi qacc qstart qend sseqid sstart send evalue pident nident mismatch gapopen"')
               return ("BLAST Results are ready!",results)
           except:
               return ("BLAST Failed to execute! Check the error below.\n" + traceback.format_exc(), "")

       # perform_blast_align
       #
       # @param1 = query_db: the fasta used for comparison
       # @param2 = blast_db: the blast database
       # @param3 = run_dir: the directory of operation
       # @param4 = run_name: for file naming operation
       # @param5 = evalue: the likely hood of randomness value
       # @param6 = hits : the number of hits returned per sequence
       # @param7 = blast_type: the type of blast to perform
       #
       # return = tuple = an operation message, a tab delimited file of hits in the subject_db
       #
       # Performs NCBI local alignment blast and returns a file containg alignments
       def perform_blast_align(query_db, blast_db, run_dir, run_name, evalue, hits, blast_type, task):
           results = os.path.join(run_dir, (run_name + '.alignments.txt'))
           try:
               if blast_type == 'blastp':
                   blast_type = resource_path("blastp.exe")
               if blast_type == 'blastn':
                   blast_type = resource_path("blastn.exe")

               os.popen(blast_type + ' -db ' + blast_db + ' -query ' + query_db + ' -out ' + results +
                         ' -num_descriptions ' + hits + ' -num_alignments ' + hits + ' -evalue ' + evalue  +' ' + task + ' -outfmt 0')
               return (("Your alignments are complete! " + results  + " Has been generated."), results)
           except:
               return ("BLAST Failed to execute. Check the below error!\n" + traceback.format_exc())



       # summarize results
       #
       # @param1 = results: a file of tab delimited blast results
```

```python
    # @param2 = % criteria: for determining an imperfect match
    # @param3 = the number: of hits returned.
    #
    # return = tuple = a message of operation a summary file of the hits and their metrics, dictionary of updated id's
    #
    # This method takes in the results file from perform_blast and summarizes the results
    def summarize_results(results, cut_off, hits):
        try:
            summary = results.replace('blast.raw.txt', 'blast.summary.tsv')

            new_id = {}
            count_below = 1
            count_above = 1
            avg_float = 1
            with open(results, 'r') as results_in:
                with open(summary, 'w+') as summary_out:
                    summary_out.write('Input ID\tUpdated ID\t% Identity\tMatch Type\tE-Value\t# Matches\t# Mismatches\t# Gaps\tReference Start\tReference End\n')
                    for line in results_in:
                        line = line.strip('\n')
                        metrics = line.split('\t')
                        if (float(metrics[8]) < float(cut_off)):
                            count_below +=1
                            continue
                        else:
                            if (float(metrics[8]) == 100.0):
                                match_type="Perfect"
                            elif (float(metrics[8]) > 95.00):
                                match_type="Near-perfect"
                            else:
                                match_type="Imperfect"

                            summary_out.write(metrics[1] + '\t' + metrics[4] + '\t' + metrics[8] + '\t' + match_type + '\t' + metrics[7]  + '\t' +
                                                  metrics[9] + '\t' + metrics[10] + '\t' + metrics[11] + '\t' + metrics[5] + '\t' + metrics[6] + '\n')

                            new_id[metrics[1]] = metrics[4]
                            count_above +=1
                            avg_float = avg_float + float(metrics[8])
            message = ("Summerization Complete!\n" + str(count_below) + " Sequences require updates\n" + str(count_above) + " exceeded the % match cut off for updating.\n" +
                     "Your query had a " + "{0:.2f}".format(avg_float) + " % identity to the subject\n" + "Your detailed summary is located in " + summary)
            if (len(new_id) <1):
                new_id = "There was 100 % Concordance. No Updates Required!"

            return (message, summary, new_id)
        except:
            return(("Your results could not be summerized! Check the error below.\n" + traceback.format_exc()), "", "")

    # updated_fasta
    #
    # @param1 = run_dir: the directory of operation
    # @param2 = run_name: for file naming operation
```

```
        # @param3 = dictionary of updated annotations corresponding to the user id's
        # @param4 = the user input FASTA file
        #
        # generates a fasta with only updated sequences
        def update_fasta(run_dir, run_name, new_id, query_db):
            try:
                new_fasta_name = run_name + '.updated.hits.only.fasta'
                new_fasta = os.path.join(run_dir, new_fasta_name)
                # This array records annotations that are not updated, it has no current use
                # but will be a necessary feature for future development (i.e global alignment)
                no_anno = []
                with open (query_db, 'r') as query_in:
                    with open (new_fasta, 'w+') as fasta_out:
                        updated = False
                        for line in query_in:
                            if line.startswith('>'):
                                def_line = line.strip('>')
                                def_line = def_line.strip('\n')
                                #if this sequence in the user input db has been updated, add new annotation, record this information in a dictionary
                                if new_id.has_key(def_line):
                                    fasta_out.write('>' + def_line + '|' + new_id[def_line] + '\n')
                                    updated = True
                                    continue
                                #if no match is found, simply record it in a dictionary
                                else:
                                    no_anno.append(def_line)
                                    updated = False
                            if updated:
                                fasta_out.write(textwrap.wrap(line,60))
                return (("A new FASTA file has been generated! It can be found in " + new_fasta + "\n"), new_fasta, no_anno)
            except:
                return ("Your results could not be summerized! Check the error below.\n" + traceback.format_exc(), "", "")

        #read_fasta
        #
        # @param1 = FASTA file to be read
        #
        # return = dictionary of keys(id line) and values (sequences)
        def read_fasta(fastafile):
            sequences = []
            with open(fastafile, "r") as f:
                ls = f.readlines()
                for i in ls:
                     sequences.append(i.rstrip("\n"))
            seq_id = []
            for i in sequences:
                if i[0] == ">":
                    seq_id.append(i)
            seq_id_index = []
            for i in range(len(seq_id)):
                seq_id_index.append(sequences.index(seq_id[i]))

            seq_dic = OrderedDict()
```

```python
        for i in range(len(seq_id_index)):
            if i == (len(seq_id_index) - 1):
                seq_dic[seq_id[i]] = sequences[seq_id_index[i]+1:]
            else:
                seq_dic[seq_id[i]] = sequences[seq_id_index[i]+1:seq_id_index[i+1]]

        seq_dic_2 = OrderedDict()
        for keys, values in seq_dic.items():
            seq_dic_2[keys] = "".join(values)

        return seq_dic_2

    # write_fasta
    #
    # @param1 = dictionary of sequences to be written to outfile
    #
    # Write an outfile in fasta format
    def write_fasta(dictionary, filename):
        with open(filename, "w") as outfile:
            for key, value in dictionary.items():
                outfile.write(key + "\n")
                outfile.write("\n".join(textwrap.wrap(value, 60)))
                outfile.write("\n")

    # swap_dna
    #
    # @param1 = dna sequence
    #
    # return = amino acid sequence
    def swap_dna(dnastring):
        table = {
            'ATA':'I', 'ATC':'I', 'ATT':'I', 'ATG':'M',
            'ACA':'T', 'ACC':'T', 'ACG':'T', 'ACT':'T',
            'AAC':'N', 'AAT':'N', 'AAA':'K', 'AAG':'K',
            'AGC':'S', 'AGT':'S', 'AGA':'R', 'AGG':'R',
            'CTA':'L', 'CTC':'L', 'CTG':'L', 'CTT':'L',
            'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCT':'P',
            'CAC':'H', 'CAT':'H', 'CAA':'Q', 'CAG':'Q',
            'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGT':'R',
            'GTA':'V', 'GTC':'V', 'GTG':'V', 'GTT':'V',
            'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCT':'A',
            'GAC':'D', 'GAT':'D', 'GAA':'E', 'GAG':'E',
            'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGT':'G',
            'TCA':'S', 'TCC':'S', 'TCG':'S', 'TCT':'S',
            'TTC':'F', 'TTT':'F', 'TTA':'L', 'TTG':'L',
            'TAC':'Y', 'TAT':'Y', 'TAA':'_', 'TAG':'_',
            'TGC':'C', 'TGT':'C', 'TGA':'_', 'TGG':'W',
            }
        protein = []
        end = len(dnastring) - (len(dnastring) %3) - 1
        for i in range(0,end,3):
            codon = dnastring[i:i+3]
            if codon in table:
                aminoacid = table[codon]
                protein.append(aminoacid)
            else:
                protein.append("N")
        return "".join(protein)

    # dna2aa
    #
    # @param1 = a dictionary of nucleotide sequences
    #
    # @return = a dictionary of amino acid sequences
    def dna2aa(dna_dict):
        for key, value in dna_dict.items():
             dna_dict[key] = swap_dna(value)
        return dna_dict


    def translate(fasta, run_dir):
        new_file = fasta.replace('fasta', 'aa.fasta')
        new_file = os.path.join(run_dir, new_file)
        try:
            nuc_dict = read_fasta(fasta)
            aa_dict = dna2aa(nuc_dict)
            write_fasta(aa_dict, new_file)
            return ((fasta + " Has been translated!"),new_file)
        except:
            return (fasta  + "  could not be translated! Check the error below.\n" + traceback.format_exc(), "", "")
```

Bibliography

Sanger, F., Nicklen, S., & Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences, 74(12), 5463-5467*. doi:10.1073/pnas.74.12.5463

Anderson, S. (1981). Shotgun DNA sequencing using cloned DNase I-generated fragments. *Nucl Acids Res Nucleic Acids Research, 9(13), 3015-3027*. doi:10.1093/nar/9.13.3015

Staden, R. (1979). A strategy of DNA sequencing employing computer programs. *Nucl Acids Res Nucleic Acids Research, 6(7), 2601-2610*. doi:10.1093/nar/6.7.2601

Lander, E. S. (2001, February 15). Initial sequencing and analysis of the human genome. *Nature, 409(6846), 860-921*. doi:10.1038/35057062

Heather, J. M., & Chain, B. (2016). The sequence of sequencers: The history of sequencing DNA. *Genomics, 107(1), 1-8*. doi:10.1016/j.ygeno.2015.11.003

Miller, N. A., Farrow, E. G., Gibson, M., Willig, L. K., & Twist, G. (2015). A 26-hour system of highly sensitive whole genome sequencing for emergency management of genetic diseases. *Genome Medicine Genome Med, 7(1)*. doi:10.1186/s13073-015-0221-8

An Introduction to Next-Generation Sequencing Technology [Advertisement]. (2016, May25).Illumina. Retrieved June/July, 2016, from http://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf

Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990) "Basic local alignment search tool." *J. Mol. Biol. 215:403-410*.

NCBI. The NCBI Eukaryotic Genome Annotation Pipeline. Retrieved March/April, 2016, from http://www.ncbi.nlm.nih.gov/genome/annotation_euk/process/

Hu Y, Roesel C, Flockhart I, Perkins L, Perrimon N, Mohr SE. UP-TORR: online tool for accurate and Up-to-Date annotation of RNAi Reagents. *Genetics. 2013 Sep;195(1):37-45.* doi: 10.1534/genetics.113.151340. Epub 2013 Jun 21. PubMed PMID: 23792952; PubMed Central PMCID: PMC3761311.

Sayers E. A General Introduction to the E-utilities. In: Entrez Programming Utilities Help [Internet]. Bethesda (MD): National Center for

Biotechnology Information (US); 2010-. Available from: http://www.ncbi.nlm.nih.gov/books/NBK25497/

The NCBI Handbook [Internet]. 2nd edition. Bethesda (MD): National Center for Biotechnology Information (US); 2013-. Available from: http://www.ncbi.nlm.nih.gov/books/NBK143764/

Zuo D, Mohr SE, Hu Y, Taycher E, Rolfs A, Kramer J, Williamson J, LaBaer J. PlasmID: a centralized repository for plasmid clone information and distribution. *Nucleic Acids Res. 2007 Jan;35(Database issue):D680-4*. Epub 2006 Nov 28. PubMed PMID: 17132831; PubMed Central PMCID: PMC1716714.

Harte et al. Tracking and coordinating an international curation effort for the CCDS Project. Database (Oxford). 2012 Mar 20;2012:bas008. doi: 10.1093/database/bas008. Print 2012. PubMed PMID: 22434842; PubMed Central PMCID: PMC3308164.

Pruitt et al. The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes. *Genome Res. 2009 Jul;19(7):1316-23.* doi: 10.1101/gr.080531.108. Epub 2009 Jun 4. Erratum in: Genome Res. 2009 Aug;19(8):1506. PubMed PMID: 19498102; PubMed Central PMCID: PMC2704439.

Farrell et al. Current status and new features of the Consensus Coding Sequence database. *Nucleic Acids Res. 2014 Jan;42(Database issue):D865-72*. doi: 10.1093/nar/gkt1059. Epub 2013 Nov 11. PubMed PMID: 24217909; PubMed Central PMCID: PMC3965069.

Schwartz, R.M. & Dayhoff, M.O. (1978) "Matrices for detecting distant relationships." *Atlas of Protein Sequence and Structure, vol. 5, suppl. 3.* M.O. Dayhoff (ed.), pp. 353-358, Natl. Biomed. Res. Found., Washington, DC.

Camacho C., Coulouris G., Avagyan V., Ma N., Papadopoulos J., Bealer K., & Madden T.L. (2008) "BLAST+: architecture and applications." *BMC Bioinformatics 10:421*

Pruitt et al. RefSeq: an update on mammalian reference sequences. *Nucleic Acids Res. 2013*

Tatusova T, Ciufo S, Fedorov B, O'Neill K, Tolstoy I. RefSeq microbial genomes database: new representation and annotation strategy. *Nucleic Acids Res. 2014 Jan 1;42(1):D553-9*

OC participants, Dana Farber Cancer Institute for Cancer Systems Biology, Mammalian Gene Collection, Welcome Trust Sanger Institute, Virginia G.

Piper Center for Personalized Diagnostics at Biodesign Institute of Arizona State University, DNA I.M.A.G.E Consortium. http://www.orfeomecollaboration.org. Web, Septemeber 15th, 2015.

Daniel R Zerbino, Steven P Wilder, Nathan Johnson, Thomas Juettemann and Paul R Flicek. The Ensembl Regulatory Build. *Genome Biology* 16:56. March, 2015

Lindo, S. (2013, November 07). XML vs. JSON - A Primer. Retrieved February 16, 2016, from http://www.programmableweb.com/news/xml-vs.-json-primer/how-to/2013/11/07

Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org

Brandle, G. (2016, June 16). PEP 8 -- Style Guide for Python Code. Retrieved March 27, 2016, from https://hg.python.org/peps/file/tip/pep-0008.txt

Python 2.7.12 documentation. Retrieved February 27, 2016, from https://docs.python.org/2/

PyInstaller 3.2 software. Available at http://www.pyinstaller.org/

Rappin, Noel; Dunn, Robin (March 1, 2006). wxPython in Action. Greenwich: Manning Publications. p. 552. ISBN 978-1-932394-62-7.

How to Learn wxPython. (n.d.). Retrieved June 27, 2016, from https://wiki.wxpython.org/How to Learn wxPython

Forrest, Boa. Pairwise Alignment In Python (2011). GitHub Repository https://github.com/alevchuk/pairwise-alignment-in-python/blob/master/alignment.py

P. (2016). Prestevez/dna2proteins. Retrieved August 18, 2016, from https://github.com/prestevez/dna2proteins

Enthought Python Distribution (Version 7.3) [Software]. (2012). Retrieved from http://www.enthought.com

Tools and Data. (n.d.). Retrieved August 05, 2016, from http://horfdb.dfci.harvard.edu/index.php?page=toolsdata

Simone, M. (2004). RNA interference: Learning gene knock-down from cell physiology. *Journal of Translational Medicine*, 2(1), 1479-5876. doi:10.1186/1479-5876-2-39

Kieleczawa, J. (2005). Simple Modifications of the Standard DNA Sequencing Protocol Allow for Sequencing Through siRNA Hairpins and Other Repeats. *Journal of Biomolecular Techniques* : JBT, 16(3), 220–223.

MGC Project Team. The status, quality, and expansion of the NIH full-length cDNA project: the Mammalian Gene Collection (MGC). Genome Res. 2004; 14:2121-7 and Supplemental Research Data.