**Omics Database Generator**

**Manual**

Joseph Guhlin

**1 Introduction**

This software takes genomic data, including sequence data, gene definitions, annotations, expression, and other data to compile a queryable graph-theory based database for intra- and interspecies comparisons. Primarily analysis is completed by using established tools and common assumptions to process the output of those programs. Custom data can also be entered into the database for advanced queries.

Queries can be completed by command-line, a provided web interface, or programmatically either through an API or directly to the database.

**Need for this Pipeline**

With a rapid expansion of genomic resources researchers need the ability to rapidly compare genomes and putative coding regions to existing, annotated, curated genomes and datasets in order to derive maximum information.

Genomic resources are increasing at an accelerated rate. Just as well, existing genomic resources are being updated with new assemblies and new annotations. Assemblies and annotations can differ widely between releases due to new software releases, additional sequencing, and new information regarding genetic data. Existing material may refer to an older annotation or assembly and becomes difficult to use with newer releases unless additional data mining is performed. This pipeline automates many of these steps and returns the data in a comprehensive database.

Genomic comparisons are also incredibly useful in this time of accelerated data expansion. This pipeline automates protein blast searches for homology in the same species and between species, with a focus on comparisons between other species in the search for statistically significant orthologs.

Genomic context is important for helping to validate GWAS candidates before performing additional labwork. Genomic context is also important in comparisons between species as well as deriving additional understanding of a genome. This pipeline automates the importation of Gene Ontology (GO) terms and Plant Ontology (PO) terms. The importation of Interproscan results are also handled and extrapolated to link the gene models to the GO terms as appropriate. Terms for Plant Ontology are imported if a TAIR style file is provided(only for Arabidopsis so far that I can find !!!!!). Expression data from Cufflinks output is also imported and interpreted using Pearson correlation looking for both positive and negative correlation and tissue or experimental condition specificity.

Additionally, flanking sequence tagged data is mapped onto the assemblies and affymetrix probe sets are mapped to the mRNA models. This allows rapid remapping when a new assembly is released.

This allows searches via gene model s or chromosome and base pair location numbers (in the form of Chr:Loc such as Chr2:340812 or 2:543923), and returns the nearest genes, and if there is enrichment of GO terms or PO terms, and if there is enrichment of GO/PO terms in orthologous regions of other species. Enrichment for tissue specificity or experimental condition specificity is also provided if the data is available. Mutants and affymetrix probeset data are also returned if in the nearby genomic neighborhood.

**What this Pipeline is Not**

This pipeline does not, at this time, do any gene prediction . The pipeline takes in the GFF files provided and puts on its blinders. It is theoretically possible to use GFF files from cufflinks output, or a gene prediction software, or the union or intersection of the two, and use that to look for good evidence of gene models, but I believe there are better prediction suites out there that could replicate and exceed the capabilities of this pipeline in that respect.

**Brief Overview of Data from this Pipeline**

This pipeline combines several different types of data, as well as additional analysis of this data, into a unified database. This allows some unique queries that would be difficult or impossible with only the raw files. This also allows some queries Some examples include:

* Translate Gene IDs from one annotation to another annotation
* Orthologs between species, including Blast Score Ratios (BSR)
* Genomic neighborhoods(a location on a chromosome and all nearby genes), including data such as
  + Expression enrichment for genomic neighborhoods
  + Gene Ontology / Plant Ontology term enrichment for genomic neighborhoods
  + Families and Domains via Interproscan enrichment for genomic neighborhoods
  + Similar neighborhoods in species and in out species
* Neighborhoods can also be created by co-expression and negatively correlated genes, as well as orthologs
* Possible locations of Transposon mediated mutant insertions
* Possible miRNA locations
* Ontology queries for genes and regions
* Syntenic regions based on gene models and blast searches
  + Can be used as additional evidence for Gene ID translations between annotation versions
  + Can be used with protein blast results between species as well

**2 Getting Started**

After extracting the compressed archive (.zip or .tgz) of ODG you can begin placing your data files in the *data* directory. For each assembly and annotation data you will create a directory. It is recommended to name each directory with something descriptive, such as “Arabidopsis\_thaliana\_10” or “At10” to help you in the configuration step. Each assembly should be placed into individual folders, with annotation and other related data in the same folder. ODG provides the concept of a species and a “version,” which could mean the assembly release version, or could mean strain or accession depending on your use-case. See *Fig 2.1* for an example of what a data directory’s contents can be.

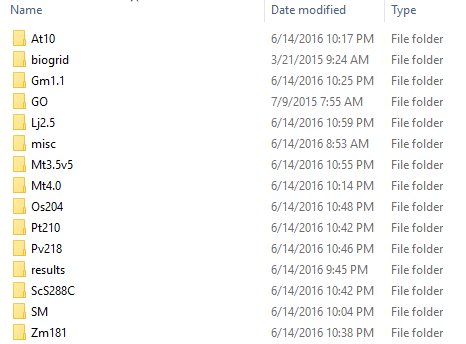


Figure 2.1. Example contents of data directory; “misc,” “biogrid,” and “GO” contain additional annotation metadata while “results” is auto-generated during later processing steps. The rest of the directories represent species.

Place all associated files for each assembly and version in their associated directories as exampled in Figure 2.2. Accepted file types and formats are:

* FASTA files for Assembly, Proteins, Transcripts, miRNA definitions, and gene sequences
* GFF3 files for genes and other features definitions, also appended FASTA assembly
* TSV – InterProScan Results, BLAST+ Results
* .hmm.tbl – HMM Results
* Pathways – Must be same format as PlantCyc
* .assoc – GAF 2.0 – Ontological Associations File
* .gtf / .fpkm\_tracking – Cufflinks Expression

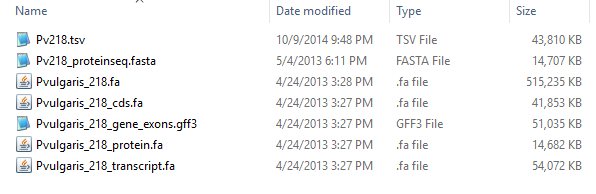


Figure 2.2. Contents of Pv218 folder, showing the assembly FASTA file (Pvulgaris\_218.fa), the CDS FASTA file (Pvulgaris\_218\_cds.fa), gene definition file (Pvulgaris\_218\_gene\_exons.gff3), protein FASTA file (Pvulgaris\_218\_protein.fa), and the InterProScan results file (Pv218.tsv).

### Global Metadata Files

##### Gene Ontology

GO/go.obo – Download the latest version of go.obo at <http://geneontology.org/page/download-ontology>

##### ENZYME

misc/enzyme.dat – Download the latest version of enzyme.dat from <ftp://ftp.expasy.org/databases/enzyme>

##### UNIPATHWAY

misc/unipathway.obo – Download the latest version of unipathway.obo from UniPathway’s website

##### Molecular Interactions

misc/mi.obo – Download the latest from <http://ontologies.berkeleybop.org/mi.obo>

### Configuring the Database

To configure the database click on start-config.bat or run bash ./config.sh and point your web browser to <http://localhost:33333/> This will give you the initial configuration screen (unless you are editing a previous configuration). It may take a moment to load depending on the size of your dataset. Figures 2.3 and 2.4 show initial configuration parameters and an example for a specific genome configuration. Remember, each genome must be in a separate folder in the data directory, this translates into separate entries for the configuration program. Once everything is configured, you must save the configuration file.

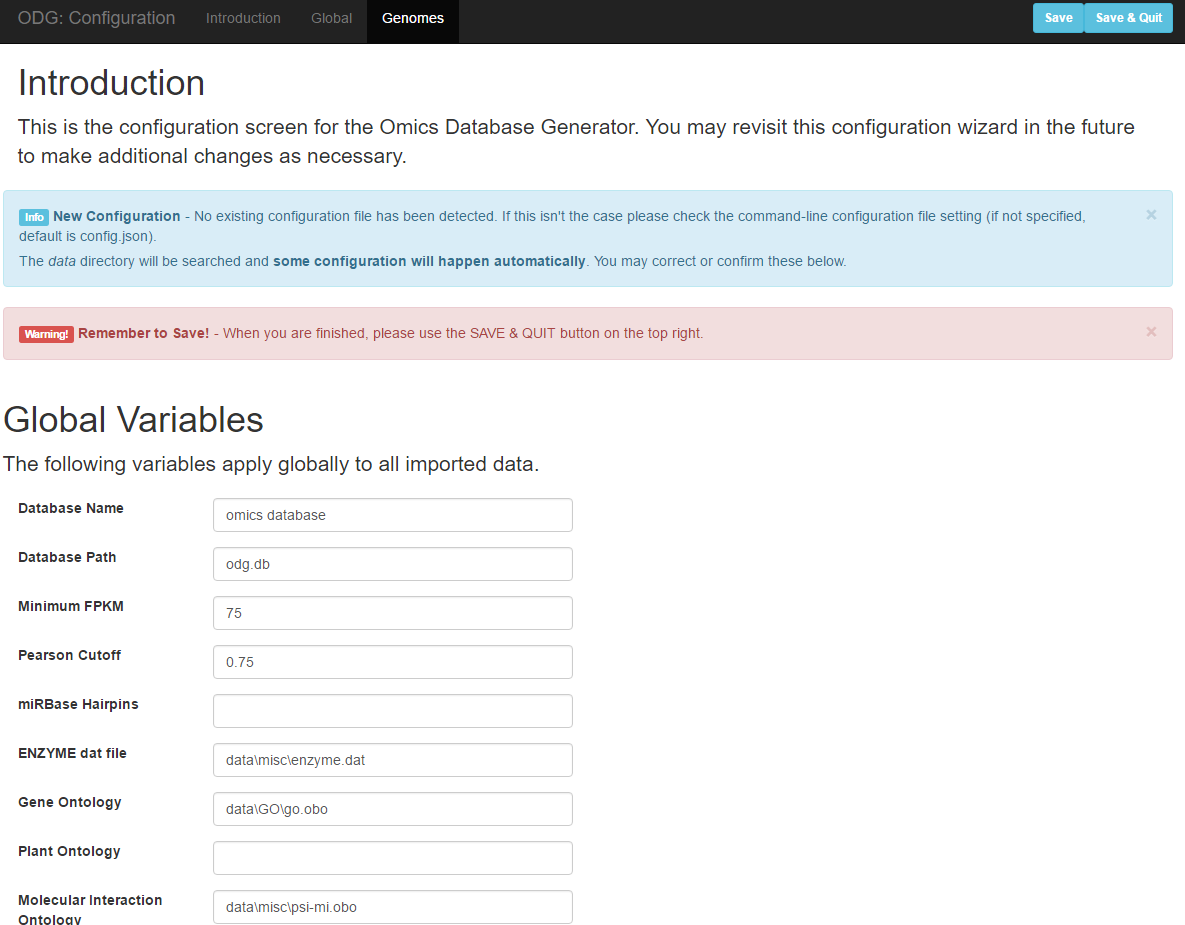


Figure 2.3. The initial configuration screen. Here you can set the database name, the path of the database, which is important if you are building multiple ODG instances, and set several other variables. If you are not using files you may leave them blank. When possible, ODG’s configuration screen tries to identify and fill in certain files when they are present.

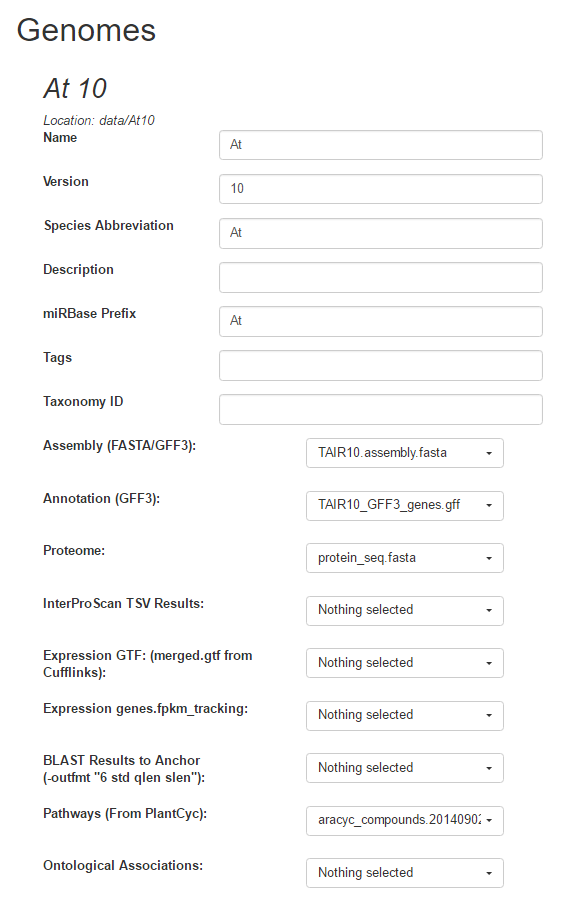


Figure 2.4. Initial screen for a genome configuration entry. When you are first editing a genome ODG will attempt to fill in as much data as possible and attempt to select the appropriate files.

### Post-Configuration Processing

#### Generating BLAST+ Scripts

The output of the entire pipeline is a graph database consisting of pseudomolecule location maps, gene model s, mRNA models, Gene Ontology and Plant Ontology terms, interproscan results, expression and correlation data, flanking sequence tags and affymetrix probeset maps, and blast result s between all of the intraspecies mRNA models and interspecies and intraspecies protein models.

The primary output of this pipeline is a comprehensive Graph database that uses Neo4j to store and access the data. A graphical user interface is also available, developed initially for the Medicago truncatula Universal Translator, but which has been adapted for a more general purpose understanding of this dataset.

This section of the manual will discuss the types of data and relationships between the data, and will not serve as a primer on how to use the optional user interface mentioned. For that please see section (!!!!).

**2.1 Basics**

**2.1.1 Nodes**

In a Graph database a node is a piece of data that serves the same function as a 'row' in a spreadsheet. A node may have incoming, outgoing, or non-directional relationships . Nodes typically have properties , which are analagous to columns in a spreadsheet, which contain additional data relevant to the node. All nodes also have a “type” which tells you what type of data the node contains, and which types of relationships are possible. For example a “gene ” node may be a “parent” of an mRNA node, which is itself a parent of an “exon ” node .

**Each node has a unique node “id” number. These ID numbers are useful to refer to nodes in queries, however they represent nothing else beyond that and are guaranteed to change between runs of the pipeline, even if no input files change.**

**Some nodes serve a purpose other than simply their type. Here are the important concepts to understand:**

* Root Nodes - Typically placeholders for other nodes to attach to. One example would be the node for a species and version, which is connected via relationships to all pseudomolecule nodes, which are themselves a type of root node.
* Anchor Nodes - More specialized root nodes. A pseudomolecule would typically be considered a root node.
* Hash Nodes - !!!! Rename - Nodes that are anchored to related genes by some arbitrary property. These are used to speed up queries. For example, genes in a 100kbp region are all on the same chromosome anchor hash, which is connected to the upstream and downstream regions with a relationship, allowing queries to look at only a subset of the data and expand as necessary. This is done to keep certain queries small and fast by limiting the dataset, while allowing expansion.

**2.1.2 Relationships**

Nodes are connected to each other by relationships , which have no analogy in spreadsheets. Relationships themselves may have properties and a type, and all relationships have a start and an end, which are both nodes. A node may have a relationship to itself, but this is not a known product of this pipeline. There are some common node types that are important to understand:

* PARENT\_OF - These relationships are directional. A gene is a PARENT\_OF an mRNA which is a PARENT\_OF an exon and CDS . PARENT\_OF relationships are common in this pipeline and are generated at several steps, especially during the GFF import.
* NEXT\_TO - Non-directional. A gene is NEXT\_TO one or more genes, typically two. A chromosome hash representing 100kbp on a psuedomolecule is typically NEXT\_TO two other chromosome hash es each representing a 100kbp region of a chromosome .

**2.1.3 Network**

The node s and the relationships between them create a network which has it's own topology . Different queries could be used to get different sections of the network , and each would have a unique topology . For example a few GO terms could be queried, along with the gene s that have them and the neighboring genes, to see how those genes would cluster , what other common properties they may have and not have. Another query that looked at ortholog pairs would see a different section of the overall network .

**The real advantage from this pipeline is not the nodes or the relationships themselves, but the connections between them. In a spreadsheet you would look up a single row and see how it compares to another row perhaps. In this database can look up a node and see all connections and all other nodes that have similar connections. Often with graph database s it is helpful to think of queries as a “shape ” that you are attempting to fit to the network.**

**2.1.4 Indexes**

This is a database so it has indexes. !!!!!

**2.2 Node Types**

This section is a list of all node types and their relationships. A section following details each relationship. This section and the following section should be used as a reference only. The advantage of this type of dataset is not the nodes or the relationships themselves, but their overall topology.

**2.2.1 main\_root**

This is the root node for everything, connected directly to species nodes. This is a good starting point for queries if no indexes work.

**2.2.2 species\_root**

These node s are root node s for a species , connected to main\_root and connected to the specific version s/accession s of a species represented in the database.

**2.2.3 root**

These are the root nodes for the species and specific version .

**2.2.4 GOTERM**

Each GOTERM node is a Gene Ontology term imported from the Obo file provided by the website.

**2.2.5 Property**

Property nodes are derived from the Gene Ontology Obo files and provide additional data and links as provided for in the Gene Ontology file. These nodes have a wide range of properties and “sub-types” which must be queried individually. This is done to allow Gene Ontology files to expand without requring additional programming of the pipeline.

**2.2.6 POTERM**

Plant Ontology Term s derived from the Plant Ontology Obo file. Plant Ontology files generate additional Property node s and relationship s as well.

**2.2.7 landmark**

Landmarks are derived from GFF files and in eukaryotic species are typically representative of chromosomes. However unanchored contigs will also create a landmark . They are used to link to landmark-hash es primary.

**2.2.8 landmark-hash**

The “ landmark-hash” type is a 100kbp region of a landmark, used to provide genomic context while limiting the total query space to speed up queries. They are linked directly to gene models, as well as to the parent landmark . The “landmark-hash ” type also has NEXT\_TO relationships spatially representative of neighboring regions.

**2.2.9 gene**

This is the gene from the GFF file. It is connected to landmark-hash nodes as well as mRNA nodes. It is also connected to other types of nodes via comparative relationship s. Gene s also have spatially relevant relationships to enable spatial queries.

**2.2.10 mRNA**

These are connected to gene node s and are conencted via comparative relationship s to other nodes.

**2.2.11 protein**

!!!!!!! Did I make these from the protein-protein comparison or are they from the GFF files for arabidopsis?

**2.2.12 exon**

Exon's are connected to mRNA nodes only.

**2.2.13 CDS**

CDS nodes are usually the exact same region as exon's and are only connected to mRNA nodes. They are left in for completeness sake.

**2.2.14 Unexpected node types**

GFF files are imported with their own types and those types are imported and linked properly to their landmarks. This can cause some additional types beyond what is described here to show. One example of this is the “chromosome” type from Arabidopsis species. These are not filtered to allow additional input from any GFF files and to allow querying. However, they are not referenced here and must be used with caution, especially in interspecies queries where only one species may have those types.

**2.3 Relationship Types**

**2.3.1 SPECIES**

**2.3.2 VERSIONRiak is a high speed key-value store database that can handle many concurrent connections.**

**2.3.3 LANDMARK**

**2.3.4 LANDMARK\_HASH**

**2.3.5 NEXT\_TO**

**2.3.6 GENE**

**2.3.7 LOCATED\_ON**

**2.3.8 PARENT\_OF**

**2.3.9 PROTEIN**

**2.3.10 HAS\_DATA**

**2.3.11 HAS\_EXPRESSION\_DATA**

**2.3.12 EXPRESSED**

**2.3.13 EXPRESSION\_CORRELATION**

**2.3.14 EXPRESSION\_NEGATIVE\_CORRELATION**

**2.3.15 HAS\_PFAM\_ENTRY**

**2.3.16 HAS\_SUPERFAMILY**

**2.3.17 HAS\_SMART\_PROPERTY**

**2.3.18 HAS\_PROSITEPATTERN**

**2.3.19 HAS\_PRINTS\_PROPERTY**

**2.3.20 HAS\_COIL**

**2.3.21 HAS\_TIGRFAM**

**2.3.22 HAS\_SIGNALP\_ENTRY**

**2.3.23 HAS\_PANTHER\_ENTRY**

**2.3.24 HAS\_PROPERTY**

**2.3.25 HAS\_GOTERM**

**2.3.26 HAS\_TERM**

**2.3.27 XREF**

From Gene Ontology

**2.3.28 is\_a**

From Gene Ontology, usually relationships between terms.

**2.3.29 intersection\_of**

From Gene Ontology

**2.3.30 regulates**

From Gene Ontology

**2.3.31 part\_of**

From Gene Ontology

**2.3.32 has\_part**

From Gene Ontology

**2.3.33 results\_in**

From Gene Ontology

**2.3.34 occurs\_in**

From Gene Ontology

**2.3.35 derives\_by\_manipulation\_from**

From Gene Ontology

**2.3.36 develops\_from**

From Gene Ontology

**2.3.37 adjacent\_to**

From Gene Ontology

**2.3.38 participates\_in**

From Gene Ontology

**2.3.39 alt\_id**

From Gene Ontology

**2.3.40 located\_in**

From Gene Ontology

**3 About the Pipeline**

**3.1 Technologies Used**

A variety of technologies, techniques, and methods are used in this pipeline. A brief overview of the technologies will allow you to better comprehend the pipeline and the final output.

**3.1.1 Graph Database**

The most important aspect of this pipeline is the final output, which is in the form of a graph database. A graph database is a database where every record is known as a node which are connected to other nodes by relationships. The graph database as opposed to a SQL database allows more unstructured data and allows complex queries to be executed.

Graph methods are used in assembly of high throughput sequencing technology. This pipeline uses graph in a more macro and meta scale.

**3.1.2 Gearman**

Gearman is an open source software that functions as a job manager. This allows tasks in the pipeline to be run in parallel, and to scale to multiple processors and computers, allowing the pipeline to run many different tasks in parallel and give faster results. This also serves to isolate computing tasks eliminating memory leaks by restarting the workers after a certain number of jobs. This pipeline does not have support for transferring files from one system to another, all files must be on all machines running the pipeline or a copy must be available via NFS. This is a point of future expansion.

At this time each task is assigned to a task group depending on the requirements and expected memory and processor load. Many tasks depend on earlier tasks to complete before they can be run, and others are independent. For example, the GFF files must be imported into the database before the FASTA files are processed, whereas the peptide databases for each data set can be generated at any time, but must occur before interspecies blastp is run.

Previously MongoDB was used which had a facility to allow files to be served from the central database when necessary, however MongoDB was not suited to the running of this pipeline in the environment it was in and Riak database was determined to be the best intermediary.

**3.1.3 Perl Data Language (PDL)**

Perl Data Language allows perl to work on very large datasets very efficiently and is used in this program

**3.1.4 Riak Database**

Riak is an open source database that can scale easily. It is used during the running of the pipeline as an intermediate database. Because gearman can run many processes in parallel Riak allows you to scale to effectively use the full capabilities of the I/O on the server it is on. Riak is not used after the graph database generation is complete, however it is part of the “complete results tar files” that contains all side effects of the pipeline.

**3.1.5 Others**

Other technologies are used but will not be explained here as it is assumed the reader is at least somewhat familiar with them or it is unnecessary for the reader to be familiar with them. Memcached is optional and speeds up the pipeline, but is unnecessary as the pipeline will fall back onto the database queries if necessary. Many CPAN modules are used and can be queried from cpan servers for further information.

Typical bioinformatics tools such as BLAST+ and Interproscan are also used and more information is available on the internet.

**4 How the Pipeline Works - A Broad Overview**

This is provided as a broad overview of how the entire pipeline works. There are a few key concepts that are necessary to understand this section.

**Workers**

To process the data more efficiently, a configurable number of workers will run in parallel the process all of the data. This speeds up the overall pipeline, but provides a few problems that must be dealt with. Not all workers are guaranteed to run on the same physical machine, so data must be accessible to all workers if they are working on that piece of data, as well as any other data requirements (GFF files often require FASTA files, for example).

**Intermediate Database**

At the time of this writing, the intermediate database is Riak, a high performance scalable database. An intermediate database is used to record and update data generated by different workers. An intermediate database allows temporary data to be generated, updated, and stored in a central location at the same time from several workers. The necessary data from the intermediate database is exported into flat files before being imported into the final database. The intermediate database is no longer used after this part.

**Final Database**

The final database is the official “output” of this pipeline. Some data is directly imported into the final database.

**4.1 Input data**

Most input sets are optional, however the more complete the data input into the system the better the results will be.

* Pseudomolecules in FASTA format
* GFF files in GFF3 files, giving gene and mRNA definitions. Exon definitions are required for tests for exon / intron understanding
* Protein models in FASTA format
* FST Mutants in FASTA format
* Affymetrix probe sets in FASTA format
* BACs in FASTA format
* !!!!! miRNA's from miRbase

**4.1.1 External dataHigh speed access to individual parts of the database**

Some datasets are not computed automatically by the pipeline. In the future they may be run automatically.

* Interproscan Results - Newest interproscan should be run with TSV output (Tested with Interproscan 5)
* Cufflinks/Cuffdiff results for Gene Expression

**4.2 Import and compute the data**

All the data is imported into an intermediate database, Riak as of the time of this writing.

* Psuedomolecules in FASTA format are imported into the database to provide
  + A centralized location of the FASTA files for workers that may be located on a separate machine
* GFF files are imported into the database, only mRNA entries are imported
  + Exons and other pieces of the GFF file are unnecessary at this time, as they are products ready for insertion into the final database.
* Protein MD5 checksums are computed from proteins found in FASTA files (!!!!! I just disabled this)
* Entire mRNA's are exported as defined by the appropriate GFF/FASTA files as necessary.
* Blast databases are created for:
  + All provided assemblies
  + mRNA sequences for the most current version
  + All protein sequence files
  + BAC sequences if provided
* Intraspecies:
  + Once created, all mRNA entries for all versions/accessions of a species are blasted(blastn) against target version
  + All FST databases are blasted against target assemblies
  + mRNA entries are blasted against newest assemblies
  + All mRNA entries are blasted against BAC databases
* Interspecies
  + All protein models are blasted against all other species protein databases

**4.3 Analyze and Import the Data**

Once all of the data is generated the next step is to analyze all of the data and import it into the Graph database. The important parts of the intermediate database are exported, and the server is disabled and archived, before this step begins.

**4.3.1 Gene Ontology / Plant Ontology database definitions**

The provided gene ontology and plant ontology definitions are imported, including all relationships.

**4.3.2 Computed MD5 Checksums**

This allows connections to interproscan and fast comparisons between mRNA.

**4.3.3 GFF Files**

GFF files are imported and proper relationships are generated. For example:

A chromosome contains a chromosome hash (RENAME??) which contains gene models. Gene models are :PARENT\_OF mRNA which are :PARENT\_OF exons and CDS entries.

**4.3.4 Protein Sequences**

Protein sequences are imported. If an mRNA entry exists for each protein sequence, no action is taken, otherwise a node is created for each protein. This is done because some input files may not have GFFs or Assemblies. MD5 checksums are also created if necessary.

**4.3.5 Protein and mRNA Checksums**

Protein and mRNA md5 checksums are imported from a separate file generated during the initial data generation process.

**4.3.6 Plant Ontology Terms**

Plant ontology terms linked to genes are imported. The file must be provided in TAIR format. So far I have only found this file as provided for Arabidopsis thaliana.

**4.3.7 Interproscan Output Files**

Interproscan output files are imported for each species if provided. Nodes are created for each property scanned, and relationships are built for all of those. The only exception are GO terms, as the GO database is already imported; relationships are built from the protein/mRNA nodes to the GO nodes.

**4.3.8 BAC FASTA Files**

Nodes are created for each entry in the BAC fasta files. This occurs so relationships can be formed between mRNA nodes to the BAC nodes based off of blast results.

**4.3.9 BLASTN mRNA vs mRNA Results**

The blastn output files are parsed to look at differences between multiple versions of a species. Results are sorted and filtered by score, coverage, and compared and recorded with other significant blast hits, if any.

**4.3.10 BLASTN mRNA vs BAC Results**

The blastn output files are parsed to look at locations of mRNA on BACs. This is also used in the later logic to differentiate between annotation versions when blastn produces an ambiguous result.

**4.3.11 BLASTP Results**

The blastp output files are parsed. Please refer to that section of the manual for more information.

**4.4 Emergent Properties**

This pipeline combines different datasets, including generated datasets. This provides some additional data not directly present in any of the original files that would be difficult to determine by other methods. Please see the appropriate section to learn about some of this data that is available.

**5 Choices/Reasoning/Etc**

**5.1 Intermediate Database**

A centralized database is required for communication from multiple workers to store data and make changes on pieces of data that may have been generated by a different worker.