User’s Manual for Generation of a Refined Pan-genome Graph and Annotation of Novel Genomes using the Refined Pan-genome Graph

1. Overview

*Background*. Pan-genome analysis begins with the determination of equivalent genes/features across a set of existing genomes from the same species. JCVI has a pan-genome pipeline which determines this gene equivalency using homology and gene context based on existing annotations of the genes in the genomes. To determine synthetic engineering for the FELIX program, novel development of an entirely new pipeline was needed to enforce consistent annotation across all genomes starting from some high-quality initial annotation. This produces a refined, consistent PGG which can be used to annotate any target genome. The JCVI pan-genome pipeline is used to generate an initial PGG but refinement of the PGG, annotation of the target genome, generation of multiple sequence alignment files for clusters and edges of the PGG, and generation and analysis of features likely to be associated with genome engineering are all generated by the FELIX postprocessing module.

*Objective*. Annotate a target genome using a refined PGG while identifying nucleotide changes that represent anomalies in the context of the PGG for a species.

*Approach.* The pan-genome FELIX module is primarily determining what is novel about a target genome compared to a set of existing genomes from the same species. This is accomplished by comparing the target genome in a consistent fashion to the consistently created Pan-Genome Graph (PGG). The PGG has two main components: nodes representing genes; and edges representing the sequence between genes, and the order and orientation of the genes in the genomes. A target genome is annotated blasting the medoid sequences for each gene cluster (node of the PGG) against the genome and then uses the Needleman-Wunsch algorithm to extend the alignment if needed. If there are conflicting blast matches, then the matches are resolved based on which matches are consistent with the structure of the PGG which encapsulates gene context across the entire pan-genome. Each segment of the genome is assigned as either a known node (gene), a known edge, a unique edge, or unannotated sequence at the end of a linear contig (only applies to draft genomes or genomes with linear elements). The target sequence of a known node or edge is efficiently compared to all known alleles for that node/edge using hashes. If the target sequence is a known allele it is not novel. Otherwise novelty is determined by length statistics or a multiple sequence alignment to compare the target sequence to the existing sequences.

*Iterative PGG Generation.* The JCVI pan-genome pipeline is used with RefSeq annotated genomes (preferably complete) or some other high-quality set of consistently annotated genomes to generate an initial. Each of these genomes is then reannotated using the PGG. Once the nodes of the PGG are mapped to each of the genomes in the pan-genome a new version of the PGG is intrinsic and then explicitly extracted. This process is iterated to stability. This ensures that each genome is consistently annotated so that missing genes are not due to inconsistent annotation.

As an example of what this consistent PGG annotation provides we used *Bacillus subtilis* subsp. *subtilis* genomes. A pan-genome with 108 complete genomes available in RefSeq was constructed (reference our paper) using the JCVI pan-genome pipeline. This pan-genome was based on the RefSeq annotation of these genomes. This initial PGG was iteratively refined until the PGG stabilized (Table 1).

**Table 1: PGG stats for *Bacillus subtilis* spp. *subtilis*.**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Initial** | **Refined** |
| **Clusters** | Size of 1 | 4,434 | 3,231 |
|  | Shared (size > 1) | 8,174 | 8,204 |
|  | # gene instances in shared | 463,311 | 487,562 |
|  | Core (95% of genomes) | 3,558 | 3,778 |
| **Edges** | Size of 1 | 7,282 | 5,479 |
|  | Shared (size > 1) | 9,755 | 9,433 |
|  | # edge instances in shared | 460,452 | 485,177 |
|  | Core (95% of genomes) | 3,230 | 3,520 |

The goal of this reannotation and iteration until stabilization was to achieve consistent annotation across all genomes in the PGG leading to a more comprehensive and cohesive PGG. While the RefSeq annotations of these genomes tends to be highly consistent many small genes are often arbitrarily called from genome to genome and even more common longer genes can occasionally be missed. There are three obvious points of improvement in the refined PGG for both the cluster and edge stats: the number of size 1 clusters/edges significantly decreased presumably due to some dubious RefSeq gene calls being eliminated and some others becoming shared with other genomes; the number of core clusters/edges significantly increased showing an improvement in the consistency of annotation across all genomes; and the number of gene instances/edge instances in clusters/edges increased indicating a more consistent annotation.

Using the refined PGG to annotate a novel target genome several features can be produced based on the comparison of the target genome genes/nodes and edges to the PGG nodes and edges. For a gene it is simple to compare whether the gene is a novel or known allele and how the length of the target allele compares to other alleles for that node and similarly for edges. A complete list of the extracted features is given in Table 2. These features or feature vector can be used as input to a classifier or just as an informative descriptor of the target genome.

**Table 2: PGG feature vector output.**

|  |  |  |
| --- | --- | --- |
|  | **FEATURE** | **DEFINITION** |
| **1** | uniq\_clus: | number of unique clusters not in the PGG (this was expected to be a major feature if a large cassette of genes was engineered into a genome such as via a plasmid) |
| **2** | uniq\_clus\_alle\_0\_25 | number of genes with unique alleles in clusters with penetrance 0-25% (we divided this feature into three penetrance ranges because engineering might be more prevalent in one of these classes such as these rarer genes) |
| **3** | uniq\_clus\_alle\_25\_75 | number of genes with unique alleles in clusters with penetrance 25-75% (much fewer genes fall into this middle range but might be manipulated for niche adaptation) |
| **4** | uniq\_clus\_alle\_75\_100 | number of genes with unique alleles in clusters with penetrance 75-100% (these highly conserved genes are less likely to be functionally modified but might be codon optimized to increase or decrease expression) |
| **5** | uniq\_edge | number of unique edges (this would capture cassette insertions but also deletions of genes and rearrangements) |
| **6** | uniq\_edge\_alle\_0\_25 | number of edges with unique alleles in clusters with penetrance 0-25% |
| **7** | uniq\_edge\_alle\_25\_75 | number of edges with unique alleles in clusters with penetrance 25-75% |
| **8** | uniq\_edge\_alle\_75\_100 | number of edges with unique alleles in clusters with penetrance 75-100% |
| **9** | short\_clus | number of genes that are shorter than expected (this might occur if a gene was targeted for functional elimination but not entirely deleted) |
| **10** | long\_clus | number of genes that are longer than expected (this might occur if an insertion was used to make the gene nonfunctional) |
| **11** | short\_edge | number of edges that are shorter than expected (edges might be shortened to remove regulatory elements) |
| **12** | long\_edge | number of edges that are longer than expected (similarly regulatory elements might be added) |
| **13** | frameshift | number of protein coding genes which appear to be frameshifted (another way to make a gene nonfunctional but could be confused with sequencing/assembly error) |
| **14** | missing\_75clus | clusters with penetrance 75% or higher which are missing (not including core) (looking for unusually missing genes) |
| **15** | missing\_75edge | edges with penetrance 75% or higher which are missing (not including core) |
| **16** | miss\_sing\_core | singleton core clusters which are missing (looking for highly unusually missing genes) |
| **17** | miss\_sing\_edge | core edges which are missing |
| **18** | total\_clus | number of (nonsingleton) shared clusters found includes core (a gross feature of the genome which is only likely to catch very large-scale engineering) |
| **19** | total\_edge | number of (nonsingleton) shared edges found includes core |
| **20** | geneANI | number of genes whose ANI is less than 90% to the cluster medoid (this should catch homologous substitution of genes from other organisms) |
| **21** | rearrange | number of rearrangements (novel core edges) (not a frequent form of engineering but would be captured with this feature) |
| **22** | wgsANI | ANI measured against all the cluster medoids (this only really measures if very large-scale codon modification has occurred but also indicates if an organism, outside of the PGG set has been used such as a closely or not so closely related species) |
| **23** | dist\_clus\_alle | indicates a gene diverged from the cluster by the distance metric |
| **24** | dist\_edge\_alle | indicates an edge diverged from the cluster by the distance metric |
| **25** | col\_clus\_alle | indicates a gene diverged from the cluster by the unique column metric |
| **26** | col\_edge\_alle | indicates an edge diverged from the cluster by the unique column metric |
| **27** | very\_short\_clus | indicates engineering to delete/knockout a gene |
| **28** | very\_long\_clus | indicates engineering to knockout a gene by insertion of an element such as a transposon |
| **29** | very\_short\_edge | indicates the deletion of an edge element such as a promoter |
| **30** | very\_long\_edge | indicates the introduction of an engineered element within the edge |
| **31** | SplitGene | captures when a gene appears to be split into two or more parts in the genome |
| **32** | gapped\_clus | indicates a cluster is either partial due to being at the end of a contig or contains a run of NNNNs |
| **33** | gapped\_edge | indicates an edge is partial due to containing a run of NNNNs |
| **34** | Insertion | sequence not in the PGG genome bounded by sequence that is – this can also include a deletion |
| **35** | Deletion | sequence in the PGG genome is not present in the target genome |
| **36** | Mutation | full length match to PGG genome but below percent identity threshold |
| **37** | Tandem duplication | overlapping matches to the PGG genome |

*Detecting Engineering.* The PGG annotation of a genome divides the genome into a largely disjoint set of clusters/genes (these sometimes have small overlaps) and the edges between clusters. This means for complete genomes that every nucleotide in the genome is assigned to some cluster or edge (for draft genomes, the incomplete edge before a sequencing gap is currently left unannotated). Currently, genes/nodes/clusters are only annotated based on homology to known genes (nodes of the PGG) when supported by the genome context of the PGG. There are two types of annotated edges: known (that is they exist in the PGG) and unique (not in the PGG). Unique edges only occur when this edge between existing PGG clusters is not in the PGG. Even though a cluster or edge in the genome being annotated exists in the PGG, this does not mean that no engineering of the sequence of that cluster or edge has taken place. Homology is used to determine the cluster identification and even partial or diverse homology can be enough to assign cluster membership. Edge membership is not even based on homology but rather on the clusters abutting the edge. Currently, for an existing cluster/edge, a limited set of features are determined based on comparison to the cluster/edges in the PGG (Table 2).

The intent of evaluating an annotated cluster or edge sequence is to determine if it is likely to have occurred due to natural evolution or due to engineering. The PGG is a representation of what is present in naturally occurring members of the species group (this is only true if the genomes comprising the PGG are all naturally occurring and not engineered). It is not enough to simply determine if a genome sequence being evaluated is identical to one that has been seen before, but rather if the divergence of the sequence is significantly different than natural evolution. Towards that end, we use two distance metrics for a sequence (cluster or edge) compared to the corresponding set of sequences in the PGG: edit distance to the closest member, and number of novel column alleles in a multiple sequence alignment (that is for the row of the novel sequence how many times is a character in that row not equal to another character in the same column). Both measures will be more informative for larger and more diverse membership of the PGG and for clusters/edges in the PGG which occur in more genomes.

These distance metrics are evaluated in comparison to the known distributions of these metrics within the PGG. For each cluster and edge, for each genome in the PGG, the distance metrics are calculated as if the genome is not in the PGG but all of the other genomes are providing the distribution and statistics such as mean, median, and standard distribution. The first measure is if the median pairwise distance of the new sequence to sequences in the PGG is less than the maximum pairwise distance within the PGG. The second measure is if the number of unique columns for the new sequence is greater than the maximum unique columns for any of the PGG sequences. We would expect this approach to flag, among other things: genes/edges shortened by deletion, genes/edges lengthened by insertion, codon optimization, heterologous substitution of a homologous gene from another species, and modifying promoter regions. We would not expect this approach to flag a small number of random mutations equivalent to natural evolution which have no impact on function of the organism. Less clear is whether a single SNP which modified an active site affecting function would be flagged. If the PGG was large and diverse enough, such a single change might still stand out as an anomaly by the novel column character metric but most likely would need a more careful functional analysis. In summary, the PGG will store distributions and statistics for all clusters/edges for both metrics and these distributions combined with the metrics for novel sequences will be used to flag outlier sequences for further analysis such as codon optimization.

*Engineering Signature Identification.* A program, filter\_anomalies.pl, extracts regions of the genome associated with nine features (Table 2, Features 23-31) plus uniq\_edge (an edge not in the PGG, Table 2, Feature 5).

Segments are extracted from the genome using the following rules:

1. If a segment was annotated as one of the designated nine features of interest it was marked for extraction.
2. If marked segments were directly adjacent or overlapping, they were combined.
3. A hundred basepairs was added to each end of combined segments for genome context (especially important for short segments such as some edges), extracted, and written to a multifasta file with detailed headers.
4. Segments for the ends of unannotated contig ends > 20bp and the entire contig for contigs with no annotation were also included.

The segments are compared using BLAST to two databases:

1. the target genome (checking for repeats/paralogs and contigs which represent alternative regions in a mixed sample), and
2. the set of genomes constituting the PGG (to verify deletions, insertions, mutations, and tandem duplications with respect to the PGG).

The filter\_anomalies.pl output is already compact and relatively easy for an expert to interpret but further filtering of the BLAST output was desired to more fully automate an expert user review reducing the amount of an expert’s time needed. Most of the false positives fall into three categories: assembly errors/shortcomings, somewhat diverged nodes/edges, and short, low coverage contigs which were probably just library/sequencing contaminants. For assembly we currently use Spades with ONT long reads for scaffolding. Essentially, Spades constructs a genome using only the Illumina reads therefore preserving the small contigs that are otherwise collapsed and discarded by miniasm or flyE. Spades scaffolds the Illumina read-based contigs into larger contig(s)/scaffold(s) by using long reads which align to the previously constructed contigs. This strategy successfully maintained GFP in dilutions up to 100:1 WT:engineered where only two reads in the long-read dataset contained GFP sequence. This assembly process produced two types of anomalies which generated spurious segments: long stretches of Ns representing gaps particularly for ribosomal operons and other large repeats, and failure to overlap the preexisting contigs so that a duplicated 54bp (the k-mer size used for the assembler) on either side of a stretch of ten Ns was introduced. Both patterns are easy to detect in the tabular BLAST output or by other methods. We also tried assembling the ONT reads first using flyE and polishing the resulting contigs using pilon. When coverage depth was high enough this appeared to eliminate most assembly-based spurious segments. When ONT coverage was lower we saw two problems: incomplete polishing where many nodes/edges in some areas of some contigs were generating excess numbers of diverged segments, and much less commonly runs of polyA which were artifactual. These issues are a little harder to capture in an automated way but still possible. We believe that assemblers can do better and have been working with some assembly groups towards this goal. We may also attempt to create our own merging of these assemblies.

To address gaps in the assemblies, code was added to pgg\_edge\_multifasta.pl and pgg\_anotate.pl to recognize when a gene or edge sequence contained a run of Ns and to count these as two new features: gapped\_clus or gapped\_edge instead of into some other diverged feature (Table 2, Features 32-33). In addition, genes/edges which abutted contig ends or gaps (runs of Ns) were flagged as possibly affected by the unassembled gap and discounted.

The segments representing unannotated contig ends and whole unannotated contigs are easy to automatically triage. Most of these sequences are just a failure of the assembler possibly due to lack of sequencing depth or length and can be quickly ignored based on BLAST matches to genomes in the PGG. Only two patterns have proven so far to be worth considering engineering: for dilutions of an engineered organism in the presence of a wild type background a small contig with the engineered insertion/deletion will be flanked by sequence matching a primary assembly contig in a localized region (in fact most assemblers output an assembly graph which should indicate that this sequence was part of that graph and was left out of the primary consensus path), and contigs whose ends overlap indicating a possibly introduced plasmid (some assemblers will also indicate this circular molecule in the assembly graph or more explicitly).

To address contaminants and poorly assembled short contigs two steps were taken. Very-low coverage (< 2x), short (< 1000bp) contigs were ignored. Unannotated ends of contigs and short whole contigs with no annotation which had full length matches to PGG genomes were ignored.

The next most common somewhat spurious segments are those which have diverged enough to be flagged but are more likely natural divergence rather than engineering. There are multiple ways to address this: first we could set a less sensitive threshold more in line with expert opinion, or we could pass these sequences, in the case of nodes/genes, to a module to assess functional/expression modification of the gene (likewise we could look for likely important changes in promoters or other regulatory elements on edges). To facilitate the speed of processing of the nonspurious segments patterns of BLAST matches consistent with deletions (this could also be done via alignment to the medoid sequence for the gene/edge, combined version of these, or extracted sequence from the best BLAST match), insertions, or more complex rearrangements could be recognized and more clearly presented to the expert (although the current tabular BLAST output was fairly easy already for an expert to interpret).

A fairly large reduction in segments needing review by an expert is achieved by simply ignoring segments with full length matches to a PGG genome above some threshold of percent identity (currently set at 99.5%). The filter\_anomalies.pl script was modified to recognize four events based on BLAST matches to PGG genomes (Table 2, Features 34-37). These four events are now automatically captured by filter\_anomalies.pl which outputs a tab delimited set of features for each event as listed in Table 3.

**Table 3: filter\_anomolies\_pl tabular output format**

|  |  |  |
| --- | --- | --- |
|  | **Attribute** | **Definition** |
| **1** | Sample | Name of sample/genome/target |
| **2** | Type | Which of the four types |
| **3** | Query ID | Name of extracted sequence which is the original contig name plus coordinates and types of flagged events |
| **4** | 5pflank | Sequence which matches the subject genome on the 5prime flank (this is empty for Mutations) |
| **5** | 3pflank | Sequence which matches the subject genome on the 3prime flank (this is empty for Mutations) |
| **6** | Deleted | Sequence which has been deleted from the target genome with respect to the subject genome (this may be nonempty for an Insertion if a deletion occurred in addition to the insertion) |
| **7** | Deletion length | Length of deleted sequence |
| **8** | Inserted | Sequence which has been inserted into the target genome with respect to the subject genome (this contains the tandem duplication unit for that event – this can also be nonempty for deletion events but will be short since a maximum of 10bp is allowed before calling it an insertion) |
| **8** | Insertion length | Length of inserted sequence |
| **10** | Subject ID | Contig ID from PGG genomes |
| **11** | % Identity | Percent identity of the match for a mutation or the average of the flanking sequence matches for the other three types |
| **12** | Subject Start | Starting coordinate for the mutation/deletion/insertion/tandem duplication with respect to the Subject genome |
| **13** | Subject End | Ending coordinate for the mutation/deletion/insertion/tandem duplication with respect to the Subject genome |
| **14** | Query Start | Starting coordinate for the mutation/deletion/insertion/tandem duplication with respect to the extracted Query sequence (contig coordinates can be derived from the Query ID) |
| **15** | Query End | Ending coordinate for the mutation/deletion/insertion/tandem duplication with respect to the extracted Query sequence (contig coordinates can be derived from the Query ID) |

*Core Region Determination.* For a target genome, core regions are determined based on the PGG. Nodes in the PGG are orthologous clusters of genes. Edges in the PGG represented adjacency of genes (contained in the clusters) in the underlying genomes. The definition of which genes are considered “core” is determined relative to a threshold criterion. We use a criterion for core such that 95% or more of the PGG genomes must contain the cluster or edge. Each core region begins with a core cluster followed by a core edge, if possible, to another core cluster to form a path in the PGG. This path is then mapped onto any genome to determine coordinates. When the core threshold is below 100% any genome may be missing a cluster (gene) or edge along this path which results in the path being broken into its remaining constituent parts.

*Pipeline Organization.* The Pan-Genome Pipeline consists of four high level steps:

1. initial pan-genome graph generation,
2. refined pan-genome graph generation,
3. multiple sequence alignment generation, and
4. target genome annotation.

The initial PGG generation takes a set of genomes downloaded from GenBank RefSeq which are run through the JCVI pan-genome pipeline, in particular PanOCT, to generate a PGG. The refined PGG generation uses the initial PGG from the previous step to annotate the same set of genomes which were used to build the initial PGG. This reannotation is used to generate a refined PGG. This process, reannotation and regeneration of the PGG, is iterated until stable. The multiple sequence alignment generation invokes Muscle for every node and edge in the refined PGG which has more than one allele. The target genome annotation uses the refined PGG to annotate the genome and determine the evidence of possible engineering.

1. Initial pan-genome graph generation
2. Component Overview

The initial pan-genome graph (PGG) generation takes a set of genomes downloaded from GenBank RefSeq which are run through the JCVI pan-genome pipeline, in particular PanOCT, to generate a PGG. There are four components to this stage:

1. downloading a set of genomes from RefSeq,
2. running the JCVI pan-genome pipeline on those genomes
3. using MASH to calculate the approximate average nucleotide identity between each pair of genomes, and
4. Using GGRaSP to determine a set of nonredundant genomes to continue with to the next step.

1. Dependencies
2. Input(s)

|  |  |
| --- | --- |
| Input | SAMPLE TEXT BioV job file |
| Input file location | **biov** folder inside a **results/batch** folder e.g. **/data-fastpool/FELIX/results/live\_engineered/batch1/biov/config.txt** |
| Description of input | A text file specifying all BioVelocity job parameters. See example in 4.3 for more details on its contents. |
| Source of input | NA, Sample Processing is the start of the pipeline |

1. Step by Step Instructions

**ftp\_download\_ncbi\_annotation.pl** -s "Salmonella enterica" --cg -w working\_dir -o gb\_dir --id\_type biosample --id\_length 100 --both --output\_prefix Sent is a typical invocation for downloading only complete genomes from RefSeq. Option -s specifies a string to look for in the organism fields of the NCBI ftp directories. Option  --cg specifies complete genomes only. Option -w specifies a location for temporary files which can be removed later. Option -o specifies where to put the downloaded files. Option --id\_type biosample specifies to use BioSample identifiers for file name prefixes. Option --id\_length specifies a maximum identifier length before truncation. Option --both specifies to download both GenBank and fasta files for the genomes. Option --output\_prefix specifies a prefix to use for non-genome files such as log files. For further options the user can use the -h option for help information.

**run\_pangenome.pl** --hierarchy\_file hierarchy\_file --no\_grid --blast\_local --panoct\_local --use\_nuc --gb\_dir gb\_dir --no\_frameshift --panoct\_args '-L 90 -i 90 -M Y -H Y -V Y -N Y -G y -c 0,50,95,100 -T' is a somewhat atypical invocation for running the pan-genome pipeline but what was used for large PGGs (more than 200 genomes). Option --hierarchy\_file specifies a file with how the genomes should be divided into batches. The format is per line L1B# (where # starts at 1 and increments by 1) followed by (genome identifier, genome identifier, ... ,genome identifier) where the recommended number of genome identifiers per batch is 20-30. The last line of the file should be: L2B1(L1B1,L1B2,...,L1B#) where # is the largest # from the L1B# lines. Option --no\_grid specifies that Blast jobs and batch jobs are not to be farmed out to a qsub grid. Option --blast\_local specifies not to farm out Blast jobs to the grid. Option --panoct\_local specifies to not farm out PanOCT jobs to the grid. Option --use\_nuc specifies to use nucleotide level Blast searches. Option --gb\_dir specifies a directory where GenBank files for the genomes can be found. Option --no\_frameshift specifies turning off frameshift/fragmented gene detection in PanOCT because the PGG refinement step should compensate for this particularly if the genomes are complete. Option --panoct\_args allows for the pass through of most PanOCT arguments not covered by things like --no\_frameshift. The usage in the invocation given here is to specify that Blast matches must be 90% length and 90% identity in order to be clustered together. This makes sense for high quality complete genomes where for the refined PGG we rely on cluster medoid sequences for annotation and this is weakened if the cluster is too diverse. For further options the user can use the -h option for help information.

**run\_multiple\_mash.pl** is used to compute an all versus all MASH estimate of pairwise Average Nucleotide Identity (ANI). Option -i specifies genome multifasta files: a string containing directory with the fasta files, the fasta file, or an ls style search string to use (this must be in "quotes"). Option -j specifies the size of the MASH sketch (recommended 10000). Option -k specifies the k-mer size for MASH (recommended 16 for bacterial genomes). Option -y specifies to only make the ANI matrix. Option -a specifies an optional suffix (.fasta is the default) to be added to genome identifiers use "" if no suffix desired. Option -o specifies the output file prefix to use. Option -M specifies the file path for the MASH executable.

**ggrasp.R** is used to select a set of representative genomes from a larger set of genomes. For the pan-genome pipeline this can be used to remove very similar and possibly clonal genomes to avoid selection bias. Option-i specifies an all versus all matrix of ANI values. Option -d specifies a value to subtract the matrix values from to turn it into a distance rather than a similarity matrix, 100 if ANI values are 0-100. Option -m specifies which hierarchical (hclust) method to use such as complete. Option -h specifies a "height" which is the distance to generate clusters below. Option -o specifies the output files prefix. Output files are the cluster medoids and a Newick formatted tree.

1. Output(s)

|  |  |
| --- | --- |
| Output | Name of output product or file |
| Output file location | Specific file location of where this output file is saved |
| Description of output | High level description |
| Next Step(s) | State whether output is used for reporting,  Link next component within document that this output feeds into if any,  Or date – this output is not currently used |

1. Refined pan-genome graph generation
2. Component Overview

The refined pan-genome graph (PGG) generation uses the initial PGG from the previous step to annotate the same set of genomes which were used to build the initial PGG. This reannotation is used to generate a refined PGG. This process, reannotation and regeneration of the PGG, is iterated until stable.

1. Dependencies
2. Input(s)
3. Step by Step Instructions

**iterate\_pgg\_graph\_grid.pl** is the top-level script for refining the PGG. The inputs are: the initial PGG, the clusters (nodes of the PGG), the sizes of the clusters, which clusters are single copy core clusters, the medoids for the clusters, the genomes which formed the PGG,  the topology (linear or circular) of the contigs in the genomes, and the combined attributes of all the genes in all of the genomes. The outputs are a refined PGG, a refined set of clusters, refined cluster sizes, a refined set of cluster medoids, and a refined set of gene attributes. Option  -pgg specifies a file containing the initial PGG. This file is typically produced by run\_pangenome.pl and is named 0\_core\_adjacency\_vector.txt. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. The columns (genomes) of the PGG must be in the same order as the genomes inputted using the -genomes option. For run\_pangenome.pl the genomes.list file is in the correct order for 0\_core\_adjacency\_vector.txt. Option -genomes specifies a file with two columns: column 1 is the genome identifier and column 2 is the path to the multifasta file containing the genome's contig sequences. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -match specifies the file containing the clusters. The first column is the cluster number starting at 1 and increasing by 1 to the total number of clusters. The following columns are for each genome, in the order of the -genomes file, and contain the gene identifier for that genome. For run\_pangenome.pl this is the matchtable.txt file. Option -weights specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. There are two different ways to specify single copy core clusters: use -paralogs to specify a run\_pangenome.pl generated set of paralogous clusters (paralogs.txt - each row has a variable number of tab delimited columns where clusters which are paralogs are listed in the columns) or directly specify the single copy core clusters using -single\_copy which expects a file with one cluster number per line. Additional options include: -project which specifies a project code if your qsub environment requires one otherwise you should specify NONE, -qsub\_queue if your qsub environment requires you to specify one otherwise NONE, -max\_grid\_jobs specifies the maximum number of grid jobs to run at one time, -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, -blast\_task species which blast task to run, -muscle\_path specifies the path for the Muscle executable, -iterations specifies the maximum number of refinement iterations to perform if the PGG does not entirely stabilize (sometimes the PGG will oscillate between two very similar graphs), -id specifies the percent cluster size with respect to the number of genomes to be a core cluster when using the -paralogs option rather than the -single\_copy option (95 recommended), and -debug which generates more messages.

iterate\_pgg\_graph\_grid.pl  calls **single\_copy\_core.pl** (if the -paralogs option is used) which determines single copy core clusters when given two files (cluster sizes and paralogs) and a percent cutoff  for what is a core cluster. The -s option specifies the cluster sizes file (a tab delimited file where the first two columns are: the cluster number and the size of the cluster). The -p option specifies the paralogs file (each row has a variable number of tab delimited columns where clusters which are paralogs are listed in the columns). The -c option specifies a percent cutoff for cluster size. The single copy core cluster numbers are output one per line to stdout.

iterate\_pgg\_graph\_grid.pl calls core\_neighbor\_finder.pl which determines a local node centric projection of the PGG given a PGG and a list of single copy core clusters. Option -v specifies the PGG file. Option -cl specifies the single copy core file. A file called core\_neighbors is written in the current working directory which contains the local node centric projection of the PGG. This is called for each iteration of refinement since the PGG is changing.

iterate\_pgg\_graph\_grid.pl calls **compute\_pgg\_graph.pl** for each genome in the PGG (or to be exact for the genomes listed in the -genomes option to iterate\_pgg\_graph\_grid.pl) for each iteration of refinement. compute\_pgg\_graph.pl when invoked by iterate\_pgg\_graph\_grid.pl runs a blast search of the cluster medoids against the specified genome and then uses that result and the PGG to annotate the specified genome by assigning a coordinate range of the genome to a cluster in the PGG. Option -genome specifies the genome multifasta file. Option -pgg specifies a file containing the PGG. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. Option -weights specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -name specifies the genome identifier. Option -reannotate lets the script know that it is being invoked for refinement of the PGG rather than for annotation of a novel genome. Other options include: -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, -blast\_task species which blast task to run, -muscle\_path specifies the path for the Muscle executable, -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), and -debug which generates more messages. This script also expects three files to be in the working directory: "genome identifier"\_topology.txt where genome identifier is the same as the -name option, single\_copy\_clusters.txt, and core\_neighbors.

compute\_pgg\_graph.pl calls medoid\_blast\_search.pl which uses Blast to search the cluster medoids against a target genome. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -genome specifies the file path of the multifasta target genome file. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -blastout specifies a file path for where the tabular blast output should be written. Other options include: -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, and -blast\_task species which blast task to run.

compute\_pgg\_graph.pl calls **pgg\_annotate.pl** which uses the PGG and the cluster medoids Blast results to annotate the target genome. Option -reannotate specifies that the target genome is one of the PGG genomes and is being reannotated. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -blast specifies the input file with tabular Blast results of the cluster medoids searched against the target genome. Option -clusters specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -genome specifies the file path of the multifasta target genome file. Option -core specifies the single copy core clusters file with one cluster number per line. Option -target specifies the target genome identifier. Option -rootname is the identifier to use as the prefix for output file names. Option -neighbors specifies the core neighbors file output by core\_neighbor\_finder.pl. Option -pgg specifies a file containing the PGG. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge.

pgg\_annotate.pl generates numerous output files some of which are used as part of iterate\_pgg\_graph.pl. Output files are of the form rootname\_something where rootname is specified by the -rootname option. The rootname\_seqs.fasta file is a multifasta file containing novel genes determined by pgg\_annotate.pl currently based on paralogy to existing clusters but this feature is currently disabled. Future work may make use of an input file of gene annotations. rootname\_wgsANI.txt contains a single floating-point number which is the length weighted mean ANI of the final set of Blast matches for annotated genes compared to the set of cluster medoids for those genes. rootname\_geneANI.txt is tab delimited with one gene annotation per line where the Blast match is < 90% identity to the cluster medoid: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string geneANI, column 4 is the gene's 5' coordinate on the contig, column 5 is the gene's 3' coordinate on the contig, column 6 is the gene's length, and column 7 is the gene's identifier. rootname\_rearrange.txt is tab delimited with one genome rearrangement per line based on seeing single copy core genes in unexpected adjacency: column 1 is the target genome identifier, column 2 is the  contig identifier, column 3 is the string rearrange, column 4 is the 5' coordinate of the interval between the two bounding single copy core genes, column 5 is the 3' coordinate of this interval, column 6 is the length of the interval, and column 7 is the virtual edge between the two bounding single copy core genes. rootname This file has one edge per line. Both orientations of the edge are output on separate lines. The edge format is (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). rootname\_attributes.txt is the tab delimited gene attributes file: column 1 is the contig identifier, column 2 is the gene's identifier, column 3 is the gene's 5' coordinate on the contig, column 4 is the gene's 3' coordinate on the contig, column 5 is the gene annotation from the cluster medoid's fasta header line, and column 6 is the target genome identifier. rootname\_attributes\_new.txt is the same type file as rootname\_attributes.txt but for new genes which are not orthologs of existing clusters - currently new genes are not annotated and so this file is empty. rootname\_uniq\_clus.txt is tab delimited one gene per line which tracks novel genes which are not orthologs of existing clusters: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string uniq\_clus, column 4 is the gene's 5' coordinate on the contig, column 5 is the gene's 3' coordinate on the contig, column 6 is the gene's length, and column 7 is the gene's identifier. rootname\_core\_clus.txt is a tab delimited file with one core region (a set of adjacent core clusters and edges) per line: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the region’s 5' coordinate on the contig, column 4 is the region's 3' coordinate on the contig, column 5 is a list of the core gene identifiers in the core region. rootname\_new\_clus.txt gives the neighborhood context of a new gene so that this gene can be combined with other new genes from other genomes into a cluster during PGG refinement based on homology and neighborhood. This feature is currently disabled as no new genes are being annotated. This file is tab delimited with one gene neighborhood per line: column 1 is the target genome identifier, column 2 is the gene identifier, column 3 is the 5' nearest existing cluster neighbor, column 4 is the 3' nearest existing cluster neighbor, column 5 is the 5' nearest core cluster neighbor, and column 6 is the 3' nearest core cluster neighbor. rootname\_match.col has one entry per line and has the same number of lines as there are existing clusters in the PGG. If an ortholog exists in the target genome for a cluster the line has the gene identifier of the ortholog corresponding to that line number otherwise a null indicator of ----------. This file will be pasted together (using the unix paste command) with other PGG genome target files to create a new clusters file for input to pgg\_edge\_multifasta.pl. rootname\_match\_new.col is similar in that it has only a gene identifier per line for all novel genes but in no particular order - currently novel genes are not being annotated and this file is not used. rootname\_pgg.col is like rootname\_match.col but for edges instead of clusters. For each edge in the existing PGG the line will contain a 1 if the edge exists in the target genome and a 0 otherwise. This file is no longer used since its functionality has been replaced by rootname\_alledges.txt and pgg\_combine\_edges.pl. rootname\_pgg\_new.col is for edges not in the existing PGG and has the edge in column 1 in the same format as the PGG and a 1 in column 2. This file is also obsolete since its functionality has been replaced by rootname\_alledges.txt and pgg\_combine\_edges.pl. rootname\_uniq\_edge.txt is used to flag novel edges which might be indicative of genome engineering: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string uniq\_edge, column 4 is the edge's 5' coordinate on the contig, column 5 is the edge's 3' coordinate on the contig, column 6 is the edge's length, and column 7 is the edge's identifier. Each line of rootname\_split\_gene.txt specifies a cluster ortholog which appears to have been spilt in the target genome due to an insertion event. Each line has information about the multiple partial blast matches for a given cluster indicating a gene has been split. This file is currently primarily for internal development.

iterate\_pgg\_graph\_grid.pl calls **pgg\_combine\_edges.pl** after all the genomes have been reannotated using the current PGG for the current iteration. When a genome is reannotated a set of edges present in that genome is generated. These edges are concatenated together and provided as input to pgg\_combine\_edges.pl as stdin. pgg\_combine\_edges.pl generates a refined PGG to stdout. To be exact the file input to pgg\_combine\_edges.pl is a list of files of edges, one file name per line, where the file name generated by compute\_pgg\_graph.pl for each genome is "genome identifier"\_alledges.txt. Each "genome identifier"\_alledges.txt file has one edge per line of the form: (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2).

iterate\_pgg\_graph\_grid.pl calls pgg\_edge\_multifasta.pl at the end of each iteration with the refined PGG and clusters generated by the reannotation. In this invocation pgg\_edge\_multifasta.pl is merely removing clusters which were not annotated for any genome and renumbering the remaining clusters to be consecutive from 1 to the number of remaining genomes. The medoids, cluster sizes, single copy core, clusters, and PGG files need to be modified to reflect this renumbering. Option-C specifies the Muscle executable path. Option -X specifies that this is simply a renumbering of clusters invocation -Q specifies the qsub queue not used in this case. Option -V specifies to strip contig version numbers. Option -s specifies the single copy clusters file. Option -B specifies the output directory for output files. Option -b specifies the directory to find and/or write multifasta and multi-alignment cluster files. Option -g specifies a file with a list of genome identifiers (column 1) and genome multifasta files (column 2) with the genomes in the same order as the columns of the cluster file and the PGG file. Option -m specifies the cluster file: the first column is the cluster number starting at 1 and increasing by 1 to the total number of clusters, and the following columns are for each genome, in the order of the genomes (-g) file, and contain the gene identifier for that genome. Option -a specifies a combined attributes file. The combined attributes file gives the gene coordinates for all genes in the combined genomes. compute\_pgg\_graph.pl outputs a "genome identifier"\_attributes.txt file which are concatenated together into a single file. This is a tab delimited file: column 1 is the contig identifier, column 2 is the gene identifier, column 3 is the coordinate of the 5' gene end, column 4 is the coordinate of the 3' gene end, column 5 is the annotation for the gene, and column 6 is the genome identifier. Option -p specifies the PGG file. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. Option -M specifies the multifasta clusters medoids file. Option -T specifies the topology file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -A specifies to compute for all genomes not just a target genome. Option -S suppresses outputting multifasta files for the clusters and edges. Option -R indicates that the medoids, cluster sizes, single copy core, clusters, and PGG files need to be modified to reflect cluster renumbering.

1. Output(s)

1. Multiple sequence alignment generation
2. Component Overview

The multiple sequence alignment generation invokes Muscle for every node and edge in the refined PGG which has more than one allele.

1. Dependencies
2. Input(s)
3. Step by Step Instructions

**pgg\_edge\_multifasta.pl** is called at this stage to generate multifasta files, multiple sequence alignment files, and statistics for the multiple sequence files for clusters and edges with more than one allele. Edges that are zero length or where the adjoining clusters overlap (negative length overlap) do not generate these files. Option -r specifies where the Rscript command is. Option -N specifies where the directory where the statistics command, summarize\_alignment.R, is. Option -P specifies the project code for qsub installations which require one - NONE should be used otherwise. Option -j specifies the maximum number of qsub jobs to submit at any one time. Option -f specifies that full multifasta files should be written to the multifasta directory specified with the -b option. This is a memory reduction method so that all genomes in the PGG do not need to be read in at the same time. Option -F specifies that the full multifasta files are available to be read in from the multifasta directory specified with the -b option and should be used if -f is specified. Option -L specifies that multifasta (only one copy of each allele) files, multiple sequence alignment files, and statistics files should be generated. Option -C specifies the Muscle executable path. Option -Q specifies the qsub queue for Muscle jobs and statistics jobs (this should be NONE for qsub installations that do not allow queue specification). Option -V specifies to strip contig version numbers. Option -s specifies the single copy clusters file. Option -B specifies the output directory for output files. Option -b specifies the directory to find and/or write multifasta and multi-alignment cluster files. Option -g specifies a file with a list of genome identifiers (column 1) and genome multifasta files (column 2) with the genomes in the same order as the columns of the cluster file and the PGG file. Option -m specifies the cluster file: the first column is the cluster number starting at 1 and increasing by 1 to the total number of clusters, and the following columns are for each genome, in the order of the genomes (-g) file, and contain the gene identifier for that genome. Option -a specifies a combined attributes file. The combined attributes file gives the gene coordinates for all genes in the combined genomes. This is a tab delimited file: column 1 is the contig identifier, column 2 is the gene identifier, column 3 is the coordinate of the 5' gene end, column 4 is the coordinate of the 3' gene end, column 5 is the annotation for the gene, and column 6 is the genome identifier. Option -p specifies the PGG file. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. Option -M specifies the multifasta clusters medoids file. Option -T specifies the topology file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -A specifies to compute for all genomes not just a target genome.

pgg\_edge\_multifasta.pl calls **summarize\_alignment.R** which takes a single argument in this invocation, a multiple sequence alignment generated by Muscle. The result is returned to stdout and is saved in an "identifier".stats file for the cluster or edge.

1. Output(s)

1. Target genome annotation
2. Component Overview

The target genome annotation uses the refined PGG to annotate the genome and determine the evidence of possible engineering.

1. Dependencies
2. Input(s)
3. Step by Step Instructions

**multi\_pgg\_annotate\_new.pl** is the top-level script for annotating a set of target genomes using the PGG. The inputs are: the initial PGG, the clusters (nodes of the PGG), the sizes of the clusters, which clusters are single copy core clusters, the medoids for the clusters, the genomes which formed the PGG,  the topology (linear or circular) of the contigs in the genomes, and the combined attributes of all the genes in all of the genomes. The outputs are a set of files for each target genome with the target genome identifier as a prefix (see output files below). Option  -pgg specifies a file containing the initial PGG. This file is typically produced by run\_pangenome.pl and is named 0\_core\_adjacency\_vector.txt. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. The columns (genomes) of the PGG must be in the same order as the genomes inputted using the -genomes option. For run\_pangenome.pl the genomes.list file is in the correct order for 0\_core\_adjacency\_vector.txt. Option -genomes specifies a file with two columns: column 1 is the genome identifier and column 2 is the path to the multifasta file containing the genome's contig sequences for all genomes in the PGG. Option -new\_genomes specifies a file with the same format as -genomes but for all the target genomes to be annotated. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig for all genomes in the PGG. Option new\_topology specifies a file with the same format as -topology but for all the target genomes to be annotated. Option -match specifies the file containing the clusters. The first column is the cluster number starting at 1 and increasing by 1 to the total number of clusters. The following columns are for each genome, in the order of the -genomes file, and contain the gene identifier for that genome. For run\_pangenome.pl this is the matchtable.txt file. Option -weights specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -single\_copy specifies the single copy core clusters with one cluster number per line. Option -attributes specifies a file of combined gene attributes for all genomes in the PGG. This file is generated in the PGG refinement step. Option -less\_memory tells the script that full multifasta files for each cluster and edge exist and are to be used rather than recreating them based on the genome files and clusters file. This causes pgg\_edge\_multifasta.pl to use much less memory which is beneficial for grid nodes with limited resources. Option -multifastadir specifies the directory where reduced multifasta files, statistics files, and multiple sequence alignment files are located as well as full multifasta files if option -less\_memory is used. Option -pggdb is a Blast database containing the genomes form the PGG. This is used to confirm likely engineering events after outliers are identified in comparison to the PGG. Additional options include: -project which specifies a project code if your qsub environment requires one otherwise you should specify NONE, -qsub\_queue if your qsub environment requires you to specify one otherwise NONE, -max\_grid\_jobs specifies the maximum number of grid jobs to run at one time, -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, -blast\_task species which blast task to run, -muscle\_path specifies the path for the Muscle executable,  -rscript\_path specifies where Rscript is installed, and -debug which generates more messages.

multi\_pgg\_annotate\_new.pl calls **compute\_pgg\_graph.pl** for each target genome in the  -new\_genomes option to multi\_pgg\_annotate\_new.pl. compute\_pgg\_graph.pl when invoked by multi\_pgg\_annotate\_new.pl runs a blast search of the cluster medoids against the specified genome and then uses that result and the PGG to annotate the specified genome by assigning a coordinate range of the genome to a cluster in the PGG. Option -genome specifies the genome multifasta file. Option -pgg specifies a file containing the PGG. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. Option -weights specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -name specifies the genome identifier. Other options include: -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, -blast\_task species which blast task to run, -muscle\_path specifies the path for the Muscle executable, -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), and -debug which generates more messages. This script also expects three files to be in the working directory: "genome identifier"\_topology.txt where genome identifier is the same as the -name option, single\_copy\_clusters.txt, and core\_neighbors.

compute\_pgg\_graph.pl calls **medoid\_blast\_search.pl** which uses Blast to search the cluster medoids against a target genome. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -genome specifies the file path of the multifasta target genome file. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -blastout specifies a file path for where the tabular blast output should be written. Other options include: -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general), -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, and -blast\_task species which blast task to run.

compute\_pgg\_graph.pl calls **pgg\_annotate.pl** which uses the PGG and the cluster medoids Blast results to annotate the target genome. Option -reannotate specifies that the target genome is one of the PGG genomes and is being reannotated. Option -topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -blast specifies the input file with tabular Blast results of the cluster medoids searched against the target genome. Option -clusters specifies a tab delimited file where the first two columns are: the cluster number and the size of the cluster. Option -medoids specifies a multifasta file with one medoid sequence per cluster with a fasta identifier of medoid\_# or centroid\_# where # is the cluster number. Option -genome specifies the file path of the multifasta target genome file. Option -core specifies the single copy core clusters file with one cluster number per line. Option -target specifies the target genome identifier. Option -rootname is the identifier to use as the prefix for output file names. Option -neighbors specifies the core neighbors file output by core\_neighbor\_finder.pl. Option -pgg specifies a file containing the PGG. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge.

pgg\_annotate.pl generates numerous output files some of which are used as part of iterate\_pgg\_graph.pl. Output files are of the form rootname\_something where rootname is specified by the -rootname option. The rootname\_seqs.fasta file is a multifasta file containing novel genes determined by pgg\_annotate.pl currently based on paralogy to existing clusters but this feature is currently disabled. Future work may make use of an input file of gene annotations. rootname\_wgsANI.txt contains a single floating-point number which is the length weighted mean ANI of the final set of Blast matches for annotated genes compared to the set of cluster medoids for those genes. rootname\_geneANI.txt is tab delimited with one gene annotation per line where the Blast match is < 90% identity to the cluster medoid: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string geneANI, column 4 is the gene's 5' coordinate on the contig, column 5 is the gene's 3' coordinate on the contig, column 6 is the gene's length, and column 7 is the gene's identifier. rootname\_rearrange.txt is tab delimited with one genome rearrangement per line based on seeing single copy core genes in unexpected adjacency: column 1 is the target genome identifier, column 2 is the  contig identifier, column 3 is the string rearrange, column 4 is the 5' coordinate of the interval between the two bounding single copy core genes, column 5 is the 3' coordinate of this interval, column 6 is the length of the interval, and column 7 is the virtual edge between the two bounding single copy core genes. rootname This file has one edge per line. Both orientations of the edge are output on separate lines. The edge format is (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). rootname\_attributes.txt is the tab delimited gene attributes file: column 1 is the contig identifier, column 2 is the gene's identifier, column 3 is the gene's 5' coordinate on the contig, column 4 is the gene's 3' coordinate on the contig, column 5 is the gene annotation from the cluster medoid's fasta header line, and column 6 is the target genome identifier. rootname\_attributes\_new.txt is the same type file as rootname\_attributes.txt but for new genes which are not orthologs of existing clusters - currently new genes are not annotated and so this file is empty. rootname\_uniq\_clus.txt is tab delimited one gene per line which tracks novel genes which are not orthologs of existing clusters: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string uniq\_clus, column 4 is the gene's 5' coordinate on the contig, column 5 is the gene's 3' coordinate on the contig, column 6 is the gene's length, and column 7 is the gene's identifier. rootname\_core\_clus.txt is a tab delimited file with one core region (a set of adjacent core clusters and edges) per line: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the region’s 5' coordinate on the contig, column 4 is the region's 3' coordinate on the contig, column 5 is a list of the core gene identifiers in the core region. rootname\_new\_clus.txt gives the neighborhood context of a new gene so that this gene can be combined with other new genes from other genomes into a cluster during PGG refinement based on homology and neighborhood. This feature is currently disabled as no new genes are being annotated. This file is tab delimited with one gene neighborhood per line: column 1 is the target genome identifier, column 2 is the gene identifier, column 3 is the 5' nearest existing cluster neighbor, column 4 is the 3' nearest existing cluster neighbor, column 5 is the 5' nearest core cluster neighbor, and column 6 is the 3' nearest core cluster neighbor. rootname\_match.col has one entry per line and has the same number of lines as there are existing clusters in the PGG. If an ortholog exists in the target genome for a cluster the line has the gene identifier of the ortholog corresponding to that line number otherwise a null indicator of ----------. This file will be pasted together (using the unix paste command) with other PGG genome target files to create a new clusters file for input to pgg\_edge\_multifasta.pl. rootname\_match\_new.col is similar in that it has only a gene identifier per line for all novel genes but in no particular order - currently novel genes are not being annotated and this file is not used. rootname\_pgg.col is like rootname\_match.col but for edges instead of clusters. For each edge in the existing PGG the line will contain a 1 if the edge exists in the target genome and a 0 otherwise. This file is no longer used since its functionality has been replaced by rootname\_alledges.txt and pgg\_combine\_edges.pl. rootname\_pgg\_new.col is for edges not in the existing PGG and has the edge in column 1 in the same format as the PGG and a 1 in column 2. This file is also obsolete since its functionality has been replaced by rootname\_alledges.txt and pgg\_combine\_edges.pl. rootname\_uniq\_edge.txt is used to flag novel edges which might be indicative of genome engineering: column 1 is the target genome identifier, column 2 is the contig identifier, column 3 is the string uniq\_edge, column 4 is the edge's 5' coordinate on the contig, column 5 is the edge's 3' coordinate on the contig, column 6 is the edge's length, and column 7 is the edge's identifier. Each line of rootname\_split\_gene.txt specifies a cluster ortholog which appears to have been spilt in the target genome due to an insertion event. Each line has information about the multiple partial blast matches for a given cluster indicating a gene has been split. This file is currently primarily for internal development.

compute\_pgg\_graph.pl calls **pgg\_edge\_multifasta.pl** at the end to determine clusters and edges which are outliers compared to the PGG and generate a most components of a feature vector to be used downstream. Option-C specifies the Muscle executable path. Option -Q specifies the qsub queue or NONE if not needed. Option -V specifies to strip contig version numbers. Option -s specifies the single copy clusters file. Option -B specifies the output directory for output files. Option -b specifies the directory to find and/or write multifasta and multi-alignment cluster files. Option -g specifies a file with a list of genome identifiers (column 1) and genome multifasta files (column 2) with the genomes in the same order as the columns of the cluster file and the PGG file. Option -m specifies the cluster file: the first column is the cluster number starting at 1 and increasing by 1 to the total number of clusters, and the following columns are for each genome, in the order of the genomes (-g) file, and contain the gene identifier for that genome. Option -a specifies a combined attributes file. The combined attributes file gives the gene coordinates for all genes in the combined genomes. compute\_pgg\_graph.pl outputs a "genome identifier"\_attributes.txt file which for a target genome is combined with the attribute file for the PGG. This is a tab delimited file: column 1 is the contig identifier, column 2 is the gene identifier, column 3 is the coordinate of the 5' gene end, column 4 is the coordinate of the 3' gene end, column 5 is the annotation for the gene, and column 6 is the genome identifier. Option -p specifies the PGG file. The PGG file is a tab delimited file with the first column being the edge name in the format (clusterA#\_[3|5],clusterB#\_[3|5]) for example (1\_3,2\_5) which means the 3' end of cluster 1 is adjacent to the 5' end of cluster 2). The PGG file is expected to symmetrically have both directions of the edge so for our previous example (2\_5,1\_3) should also be present. After the first column in the PGG file there are N additional columns with a value of either 1 if the genome contains the edge or 0 if the genome does not contain the edge. Option -M specifies the clusters medoids file. Option -T specifies the topology file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig. Option -t specifies to compute for only the specified target genome. Option -I specifies a genome identifier in the PGG to ignore when annotating a target genome from the PGG which must be given a different identifier than the one in the PGG. Option -S suppresses outputting multifasta files for the clusters and edges.

multi\_pgg\_annotate\_new.pl calls **filter\_anomalies.pl** . Option -PGG\_topology specifies a file with three columns: column 1 is the genome identifier, column 2 is the contig identifier from the multifasta genome file, and column 3 is either linear or circular depending on what type of contig for all of the target genomes in the PGG. Option -genomes specifies a file with one line per target genome and four columns: column 1 is the genome identifier, column 2 is the path to the genome multifasta file, column 3 is the topology file for the target genome (genome identifier\_topology.txt), and column 4 is the file of outliers/anomalies found in comparison to the PGG (genome identifier\_anomalies.txt). Option -engdb  specifies a Blast nucleotide database of known engineered genes/sequences (currently not used), option -nrdb specifies the nrdb nonredundant nucleotide Blast database (currently not used), and option -pggdb specifies a Blast nucleotide database of the genomes in the PGG. Other options include: -bin\_directory which specifies where the programs/scripts for PGG analysis are such as this one, -blast\_directory which specifies where the blast executables are, -ld\_load\_directory which species a directory which needs to be exported in order to find libraries for the blast executables, -blast\_task species which blast task to run, and -strip\_version does not take an argument and directs the program to strip version numbers from contig identifiers (recommended in general).

1. Output(s)