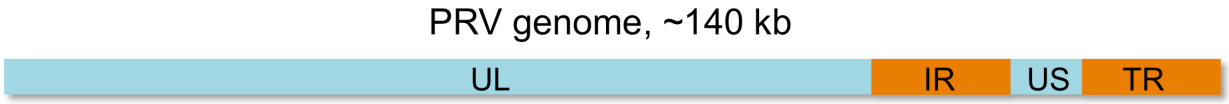
Overview & Table of Contents:



(link to top)

Part I: Pre-Assembly Processing

1. Convert Illumina output to standard format

This script (fq_all2std.pl) converted the output of an Illumina® sequencer, which is a FASTQ file using Illumina®-encoded base qualities, into a standard FASTQ format using Sanger-encoding of base qualities. It also removed the duplicated identifier line in each read, making the file size more manageable.

```
fq_all2std.pl  illumina2std
```

2. Remove adaptor contaminants, if any

As part of the Illumina® library preparation, adaptors were ligated onto fragments of sheared genomic DNA and then size-selected on a gel. If the size selection includes fragments that are too small, the resulting library will include short inserts where the sequencing reads include both genomic DNA and adaptor sequence from the far side. This script (fastx_clipper) removed any recognizable Illumina adaptor sequence. (Oligonucleotide sequence © 2007-2009 Illumina, Inc. All rights reserved.) It is part of the FASTX-Toolkit[1] (http://hannonlab.cshl.edu/fastx_toolkit/).

```
fastx_clipper -Q 33  -l  20 -a  AGATCGGAAGAGCTCGTATGCCGTCTTCTGCTTG
```

3. Remove host genome contamination

Since viruses are grown in host cells, any preparation of viral DNA can potentially include host DNA contamination. All pseudorabies virus (PRV) strains were grown in pig kidney cells. Therefore we used the Bowtie program[2] (http://bowtie-bio.sourceforge.net/index.shtml) to index the *Sus scrofa* pig genome (the 1-20-2010 release). We then checked all our sequencing reads against this host genome using Bowtie's alignment function. Finally, we extracted the identifiers of any sequence reads that matched the host genome (awk script) and removed these reads from the file used for subsequent steps (fastq-filter_extract_reads.pl script). Although we report these scripts as used, the Bowtie program now includes an option to report all unaligned sequences into a FASTQ file (--un). This would obviate the need for the last two (awk and fastq-filter_extract_reads.pl) script steps.

```
# Index the host genome:
bowtie-build
# Check for any reads that match the host genome:
bowtie
# Make a list of reads that match the host genome (IDs only):
awk  '{print $1}' filename.map
# Find and remove these sequence reads from the FASTQ file:
fastq-filter_extract_reads.pl  1 > [hostmatches.fastq]  2 >
[clean_reads.fastq]
# Newer option of Bowtie, to align to host genome and output unaligned reads:
bowtie --un
```

4. Filter out mononucleotide reads

Sequencing reads that are only one nucleotide can occur as a technical artifact, and complicate de novo assembly because of the frequent but shorter monomer repeats found in PRV strains. We used a short script (fastq-filter_mono_nucleotide.pl) to filter these out.

```
fastq-filter_mono_nucleotide.pl
```

5. Combine reads, if relevant

If two flowcell lanes were used to sequence the same PRV strain, we concatenated sequence data from the two sequencing lanes.

```
cat
```

6. Quality & length trimming

De novo assembly is improved by removing poor-quality base calls from the end of short-sequence reads produced by Illumina®. We used two scripts to achieve this. First, we used an adapted version of the quality-trimming script (TQSfastq.py) that is packaged with the SSAKE[3] de novo assembly software. With this script we modified the parameters for quality threshold (T) and the number of consecutive bases above threshold (C). The second script (fastx_trimmer) is a quality-insensitive length trimmer from the FASTX-Toolkit[1] (http://hannonlab.cshl.edu/fastx_toolkit/), which truncates all sequences at a specified length (L). These two scripts were used to generate four versions of each FASTQ file.

```
TQSfastq.py –q
TQSfastq.py  –q –T 20 –C 25
fastx_trimmer –L 41
fastx_trimmer –L 51
```

(link to top)

Part II: De novo Assembly & Genome Finishing

1. De novo assembly

The SSAKE[3] de novo assembler was used to join the short, single-end Illumina® reads into longer blocks of sequence, or contigs. Each of the four FASTQ files produced above was assembled by SSAKE under two independent conditions. First the default settings of SSAKE were used. Second the trim option (T) was called, which trims a specified number of bases from the end of a contig when all possible other joins have been exhausted. Using these two options and the four input files with varying quality & length filters, we obtained a total of eight SSAKE assemblies for each viral strain genome. Each SSAKE assembly is a multi-line FASTA file containing all newly produced contigs, varying in size from approximately 0.1-50 kb.

```
SSAKE
SSAKE –T 2
```

2. Contig merging (joining)

The Staden[4,5] DNA sequence analysis package (http://staden.sourceforge.net/) was used to further assemble the long sequence contigs produced by SSAKE[3]. Output from all eight SSAKE assemblies was imported using Staden's pregap program, which randomized their order and removed exact duplicates. These were passed to a gap4 database within the Staden program, and then processed for automated contig joining via the menu command "normal shotgun assembly". Additional contig joins were found using the menu command "find internal joins". The gap4 parameters below increase the allowed database size and number of input sequences.

```
pregap      4
gap4 -maxdb 1000000 -maxseq 8000000
```

3. Ordering of contigs and final assembly

The larger contigs produced by Staden's gap4[4,5] were exported ("save consensus" menu command) and compared to the previous PRV reference genome (NC_006151) using NCBI's blast2seq[6] (http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi). The "discontinuous megablast" option found the most homology between PRV sequences at this step. We changed several blast2seq parameters (listed below) from their default settings, to optimize matching. The blast2seq output allowed us to place the contigs in a likely order along the genome, and revealed potential bad joins in the contigs. Incorrect de novo assembly can occur at sites where the same 20 base pairs appear twice in the PRV genome. These small overlaps are sufficient for a join in Staden's gap4 program (Step 2 above) but were not supported by RFLP analysis. Additional contig adjustments, both breaks and joins, were made in Staden's gap4 to create the final assembly. Final genome assemblies were further improved by PCR validation and repeat expansion (Part III, Step 3 below), and verified by RFLP analysis.

```
discontinuous megablast
Match/Mismatch Scores = 1,-1
Gap Costs = Existence: 2  Extension: 1
Filter = OFF low complexity regions
Mask = OFF mask for lookup table only
```

4. Establish large terminal repeat

Final contig joining and curation in Staden's gap4[4,5] created PRV genomes with a unique long region (UL), a large internal inverted repeat (IR), a unique short region (US), and the beginning of the large terminal repeat (TR). We used literature references and homology searching to identify the boundaries of IR and the beginning of TR. Using a standard sequence manipulation package (e.g. MacVector®, http://www.macvector.com/), we copied the IR fragment, reverse complemented it, and merged it with the beginning of TR to create a full TR region. We queried the unused Staden contigs to search for alternative contigs that could have contributed to TR (for instance, subtle differences in short sequence repeats [SSRs] within IR and TR), but were unable to validate any such difference by PCR. The final genome containing all regions (UL-IR-US-TR) was used for all subsequent steps.

(link to top)

Part III: Quality Assessment of New Genomes

1. Test alignment to viral genome

In this step, the Bowtie[2] software package (http://bowtie-bio.sourceforge.net/index.shtml) was used to index the newly assembled viral genome and then to examine how well the Illumina® sequence data support the finished genome. An option (--best) within Bowtie was used to reduce strand bias in alignment of sequence reads, which is especially relevant due to the presence of the direct repeats (IR and TR) in the finished genomes. Another option (--sam) sent the output into .sam format. Next a script (view) from the SAMtools[7] software package (http://samtools.sourceforge.net/) was used to convert the .sam (-S) format to .bam (-b) format. Another SAMtools script (sort) puts the sequence reads of the .bam file in order along the

genome, for faster memory access during subsequent steps. We used a third SAMtools script (pileup) to determine how many sequence reads supported or covered each base of indexed genome (a "pileup" of the data). We used option (-f) to indicate the reference genome (.fasta format) and (-c) to report the consensus sequence from the sorted alignment file (.bam format). One final script (sampileup2wig.awk) converted this pileup to a wiggle graph (.wig) format, which is viewable in a genome browser such as IGB[8] (http://bioviz.org/igb/). We used the option (-v) to supply a name for this track as it appears in the genome browser.

```
# Index the viral genome:
bowtie-build
# Align reads to the new viral genome:
bowtie --best --sam
# Convert from .sam to .bam format:
samtools  view  -b  -S
# Sort order:
samtools  sort
# Generate a pileup of the data:
samtools  pileup –f –c
# Create a wiggle (.wig) graph to view in IGB or IGV:
awk  -f  sampileup2wig.awk  -v
```

2. Check for polymorphisms in the genome

The numerous sequence reads aligned to the genome in the above steps can be used to screen for any polymorphic base calls (also called single nucleotide polymorphisms, or SNPs). We used a SAMtools[7] script (varFilter) to check for polymorphic base calls. We set the depth (–D) to 40,000 to accomodate the deep sequence coverage observed across the viral genome. Another script (samsnp2bed.awk) converted the varFilter output to a file (.bed) suitable for display in a genome browser such as IGB (http://bioviz.org/igb/). We used the option (-v) to supply a name for this track as it appears in the genome browser. A final script (pileuptools.py) extracted statistics from the varFilter output, e.g. the number of sequence reads at these sites and percent support for the primary vs. secondary base call.

```
# Detect polymorphic bases:
samtools.pl  varFilter  -D  40000
# Produce a file of polymorphic bases suitable for viewing in IGB or IGV:
awk  -f  samsnp2bed.awk  -v
# Produce statistics about these polymorphic bases:
pileuptools.py snp_summary
```

3. Coverage Adjusted Perfect Repeat Expansion (CAPRE)

The initial pileup and wiggle graphs revealed several areas of extreme sequence coverage depth which coincided with perfect short sequence repeats (SSRs). The CAPRE script was used to estimate SSR length at several such sites in each PRV genome. First, bioinformatic tools available through the Galaxy[9] project (http://galaxy.psu.edu/) were used to determine the median sequence coverage from each genome's pileup file ("Summary Statistics" in Galaxy, using pileup file from Step 1 above). A tab-delimited intervals file of regions for CAPRE expansion was created for each genome. These regions were required to coincide with a perfect SSR (see accompanying manuscript for details of SSR mapping), to have coverage depth more than two

standard deviations above the median ("Pileup-to-Interval" in Galaxy, with depth cutoff), and to be intergenic (checked manually). An additional tab-delimited file of median coverage depth for a given G/C content (as determined for 10-mers tiled across the genome) was also produced for each genome. This script (repeat_expand.py) used the genome sequence, the intervals for expansion file, the G/C coverage depth file, and the sorted (.bam) file of sequence coverage along the genome (Step 1 above) to generate its output. The script estimated an appropriate SSR length at each CAPRE interval based on the expected coverage depth for its G/C content versus the actual observed coverage depth. The expanded SSR was inserted into the genome at the correct location, resulting in a finished genome (.fasta) file that was used to re-test alignment and coverage depth (Step 1 above).

```
repeat_expand.py
```

(link to top)

Part IV: Annotation & Genome Comparison

1. Migrate annotations from a reference genome to each new genome

This step uses homology between new and known genomes to annotate the location of coding sequences. For PRV genomes in this report, we extracted all genes and coding sequences (CDS regions) from the GenBank record of the mosaic reference genome NC_006151 (http://www.ncbi.nlm.nih.gov/nuccore/NC_006151). These annotations were converted to the general feature format (.gff) using the Readseq[10] biological sequence conversion tool (http://www.ebi.ac.uk/cgi-bin/readseq.cgi). This format is convenient for storing genome annotations, because .gff files can be directly loaded into genome browsers such as IGB[8] (http://bioviz.org/igb/) to display gene and coding sequence regions spatially along the genome. Next, a migration script (migrate_annotations.pl) was used to compare the original and new genome sequences, and locate homologous regions on the new genome for each sequence feature on the original genome. Because this mapping is based on homology, minor truncations can occur if nucleotide differences, insertions, or deletions (indels) occur at the edges of the homologous region. The script alerts users to this type of error if it fails to detect a start or stop codon in a coding sequence. These issues were corrected by manual inspection of the homology-based alignment (e.g. using NCBI's blast2seq, as above) and adjustment of the appropriate genome position in the resulting .gff file.

```
migrate_annotations.pl
```

2. Detection of DNA conservation across the genome

The mVISTA[11] program (http://genome.lbl.gov/vista/mvista/about.shtml) was used to create a multiple-sequence DNA alignment of the full-length genomes being compared. We used the LAGAN option with no repeat-masking for these comparisons. The genome annotations produced above can be re-formatted for upload into mVISTA (see mVISTA site for format description), to provide a view of genes or coding regions along with the DNA conservation.

3. Extracting DNA and AA sequences for each coding region

To determine which DNA sequence differences fall into coding sequences, and which subset of these affect the protein coding sequences, we first extracted the DNA and amino acid (AA) sequence for each coding region of the genome. This script (translate_cds.pl) extracted the sequence of coding regions determined in step 1 above, and reported them to a new file. By default, this script translates the defined coding sequence and reports an AA sequence. An option (-notrans) allowed for extraction of the DNA sequence within these boundaries as well. If the genome contained coding sequences that are spliced, then another option (-id) specified an identifier in the .gff file that was used to join spliced segments together.

```
# To obtain AA sequences from a genome file and its file of annotated coding sequence locations:
translate_cds.pl
# To obtain DNA sequences for the same coding regions:
translate_cds.pl – notrans
```

4. Comparison of silent and non-synonymous differences in coding regions

Once DNA and AA sequences of each coding region have been extracted for each PRV strain genome, these can be compared between strains using another script (blast_compare.pl). This script uses BLAST[6] to align any two DNA or AA sequences (e.g. reference and new), and then reports all differences between the two sequences using a standard format (reference residue-position-new residue, e.g. A100T or C200R). Insertions and deletions were reported using a dash, with the position determined on the reference sequence. An option (-type) allowed the user to specify whether DNA or AA sequences are being compared. By default, this script reported a BLAST alignment of the two sequences, which provided a visual display of differences. An option (-diff) produced a text list of the differences between two strains (e.g. UL42: E331A), for each coding sequence in the genome. This list was used as input for the next step.

```
blast_compare.pl
```

5. Determination of unique vs. shared differences between multiple sequences

When comparing two new viral genomes to a reference genome sequence, we found it useful to segregate the observed differences into those that were unique to one strain, vs. those that were shared with another strain. This script (change_comm.pl) uses the list of differences between each pair of genomes (produced in Step 4 above), to sort differences into unique vs. shared groups. Based on the input files "strain 1 differences from reference" and "strain 2 differences from reference", the default output of this script was to sequentially report differences unique to strain 1, followed by differences unique to strain 2, followed by differences from the reference that are shared by strains 1 and 2.

```
change_comm.pl
```

(link to top)

Addendum: Relevant Websites and Lists of Scripts Used

Web links to open-source software available online:

- FASTX-Toolkit[1] http://hannonlab.cshl.edu/fastx_toolkit/
- Bowtie[2] http://Bowtie-bio.sourceforge.net/index.shtml
- SSAKE[3] http://www.bcgsc.ca/platform/bioinfo/software/ssake
- SAMtools[7] http://samtools.sourceforge.net/
- Staden[4,5] (gap4) http://staden.sourceforge.net/
- Integrated Genome Browser[8] (IGB) http://bioviz.org/igb/
- Integrative Genomics Viewer[12] (IGV)
  http://www.broadinstitute.org/software/igv/home

Links for web-based analysis:

- blast2seq[6] http://www.ncbi.nlm.nih.gov/blast/bl2seq/wblast2.cgi
- Galaxy[9] http://galaxy.psu.edu/
- Readseq[10] http://www.ebi.ac.uk/cgi-bin/readseq.cgi
- mVista[11] http://genome.lbl.gov/vista/mvista/about.shtml

Scripts written or adapted at Princeton & available via Downloads link at left:

- fq_all2std.pl
- fastq-filter_extract_reads.pl
- fastq-filter_mono_nucleotide.pl
- TQSfastq.py
- sampileup2wig.awk
- samsnp2bed.awk
- repeat_expand.py
- pileuptools.py
- migrate_annotations.pl
- translate_cds.pl
- blast_compare.pl
- change_comm.pl

(link to top)

Credits:

These script instructions relate to the submitted manuscript, "A wide extent of inter-strain diversity in virulent and vaccine strains of alpha-herpesviruses", by M.L. Szpara, Y. R. Tafuri, L. Parsons, S. R. Shamim, K. J. Verstrepen, M. Legendre, and L. W. Enquist.

Current contributions

Script descriptions and instructions for use by Moriah L. Szpara (mszpara at princeton.edu).
Custom scripts written by Lance Parsons (lparsons at princeton.edu).
Website assembled by Peter Koppstein (biocomp-group at genomics.princeton.edu).
All work was done in the laboratory of L. W. Enquist (http://www.molbio1.princeton.edu/labs/enquist/) in the Department of Molecular Biology (http://www.molbio.princeton.edu/), and in the Lewis-Sigler Institute for Integrative Genomics (http://www.princeton.edu/genomics/) at Princeton University (http://www.princeton.edu).

Published and pre-existing resources:

[1] Hannon, G. J. et al. FASTX-Toolkit: FASTQ/A short-reads pre-processing tools. http://hannonlab.cshl.edu/fastx_toolkit/
[2] Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol 10, R25 (2009).
[3] Warren, R. L., Sutton, G. G., Jones, S. J. & Holt, R. A. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics* **23**, 500-501 (2007).
[4] Staden, R. The Staden sequence analysis package. *Mol Biotechnol* **5**, 233-241 (1996).
[5] Staden, R., Judge, D. P. & Bonfield, J. K. Sequence assembly and finishing methods. *Methods Biochem Anal* **43**, 303-322 (2001).
[6] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J Mol Biol* **215**, 403-410 (1990).
[7] Li, H. *et al.* The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**, 2078-2079 (2009).
[8] Nicol, J. W., Helt, G. A., Blanchard, S. G., Jr., Raja, A. & Loraine, A. E. The Integrated Genome Browser: free software for distribution and exploration of genome-scale datasets. *Bioinformatics* **25**, 2730-2731 (2009).
[9] Blankenberg, D. *et al.* Galaxy: a web-based genome analysis tool for experimentalists. *Curr Protoc Mol Biol* **Chapter 19**, Unit 19 10 11-21 (2010).
[10] Gilbert, D. G. Readseq -- biosequence conversion tool. http://iubio.bio.indiana.edu/soft/molbio/readseq/java/ (2010).
[11] Frazer, K. A., Pachter, L., Poliakov, A., Rubin, E. M. & Dubchak, I. VISTA: computational tools for comparative genomics. *Nucleic Acids Res* **32**, W273-279 (2004).
[12] Robinson, J. T. *et al.* Integrative genomics viewer. *Nat Biotechnol* **29**, 24-26 (2011).