File S1

INSTRUCTIONS - ASSEMBLY

#FILES available from https://github.com/justin-lack/Drosophila-Genome-Nexus/
round1_assembly.pl #execute the first round of mapping

round2_assembly.pl #executes the second round of mapping

VCF_filter.pl #filters SNP calls from the first round; requires >=75% of the reads at a position to match the alternate call, other wise the SNP is discarded; should be placed in the same directory as the read files and the round_1_assembly.pl script

IndelShift.pl #Removes indels added following the first round

VCF_to_Seq_Haploid.pl AND VCF_to_Seq_Diploid.pl #Produces the final
unformatted sequence files (one file for each chromosome arm),
filtering a specified number of bases around indels, implementing a
Q-value cutoff of 75 and the same 75% threshold implemented in VCF_filter.pl for alternate read proportions

#This pipeline consists of two rounds of mapping. First round maps to the publicly available D. melanogaster reference genome (or any other reference genome in fasta format), and the second round maps to a reference genome that has been updated with the SNPs and INDELS called from the first round. The original reference should have each chromosome renamed numerically in the order they are in in the reference fasta file (i.e., >1, >2, >3, etc.). This is required by the AlternateReferenceMaker module of GATK (I don't know why, but it automatically renames every chromosome in numerical order in the new reference that it creates). It's best to simply do this before you begin, and keep a record of the order of chromosomes in the file. You'll need this info for the VCF_to_Seq.pl script, because it will convert back to actual chromosome arm names when it produces the final sequence files.

#To run the assembly pipeline and produce a final all-sites VCF file, you simply need to have the two perl scripts that run each round (round1_assembly.pl and round2_assembly.pl) and the VCF_filter in a single folder with the fastq read files you want to assemble. Each instance of the pipeline should be run in its own folder containing all of the fastq input files. List the unique identifiers for your fastq files in the @FastqFile array at the top of round1_assembly.pl and add the other necessary info (i.e., paths to programs, reference name/path, etc.) and execute round1_assembly.pl. This will produce the alternate reference to be used in the second round, as well as retain the first round *realign.bam file and SNP and INDEL calls.

#If everything looks satisfactory from the first round, add the appropriate information to the top of the round2_assembly.pl script and execute this script. This will map again, but to the updated reference instead of the original reference. This will output an all-sites VCF that will serve as the input for two additional perl scripts (IndelShift.pl and VCF_to_Seq.pl), as well as a second round *realign2.bam file.

#After the second round of assembly, the all-sites VCF (from the second round) and INDEL calls (from the first round) serve as input for the IndelShift.pl script, which removes the indels we inserted following the first round. The shifted VCF output of this script then serves as input for VCF_to_Seq.pl, which outputs unformatted sequence files for each chromosome arm, filtered for a Q-value of 75, 75% alternate read proportion, and 3 bases around indels.

#EXPLANATION OF COMMANDS

FIRST ROUND:

#Assembly commands

#Uncomment next 10 lines if reference genome has not been indexed for BWA, Stampy, or GATK.

THESE ARE THE INDEXING STEPS FOR THE REFERENCE FOR ALL STEPS IN THE FIRST ROUND. THIS ONLY NEED TO BE DONE ONCE FOR REFERENCE IN THE FIRST ROUND

```
$cmd = "bwa index -a bwtsw " . $reference . ".fasta";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "CreateSequenceDictionary.jar REFERENCE=" . $reference . ".fasta OUTPUT=" . $reference . ".dict";
system($cmd);
$cmd = "samtools faidx " . $reference . ".fasta";
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -G " . $reference . " " . $reference . ".fasta";
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -g " . $reference . " -H " . $reference;
system($cmd);
```

#Assembly commands

```
for ($i = 0; $i < @FastqFile; $i++){
$cmd = "bwa aln " . $reference . ".fasta " . $FastqFile[$i] . "_1.fastq > " . $FastqFile[$i] .
"_1.sai"; #MAPPING OF READ FILE 1
system($cmd);
$cmd = "bwa aln " . $reference . ".fasta " . $FastqFile[$i] . "_2.fastq > " . $FastqFile[$i] .
"_2.sai"; #MAPPING OF READ FILE 2
system($cmd);
$cmd = "bwa sampe -P " . $reference . ".fasta " . $FastqFile[$i] . "_1.sai " . $FastqFile[$i] .
"_2.sai " . $FastqFile[$i] . "_1.fastq " . $FastqFile[$i] . "_2.fastq > " . $FastqFile[$i] . ".sam";
#COMBINING MAPPED FILES 1 AND 2
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_1.sai"; #REMOVES *.sai FILE 1
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_2.sai"; #REMOVES *.sai FILE 2
system($cmd);
$cmd = "samtools view -bS " . $FastqFile[$i] . ".sam > " . $FastqFile[$i] . ".bam";
#CONVERTS TO BAM FORMAT FOR Stampy INPUT
system($cmd);
$cmd = "rm " . $FastqFile[$i] . ".sam"; #REMOVES SAM FILE
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -g " . $reference . " -h " . $reference . " --
bamkeepgoodreads -M " . $FastqFile[$i] . ".bam -o " . $FastqFile[$i] . "_remapped.sam";
#Stampy MAPPING STEP. THIS IS A RELATIVELY LONG STEP (AS LONG AS 15 HOURS
ON SOME OF THE HIGHEST COVERAGE DPGP2 GENOMES)
system($cmd);
$cmd = "rm " . $FastqFile[$i] . ".bam"; #REMOVES BAM FILE
system($cmd);
$cmd = "samtools view -bS -q 20 " . $FastqFile[$i] . "_remapped.sam > " . $FastqFile[$i] .
"_remapped.bam"; #FILTERS BY MAPPING QUALITY 20 AND CONVERTS TO BAM
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_remapped.sam"; #REMOVES SAM FILE
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "CleanSam.jar INPUT=" . $FastqFile[$i] .
"_remapped.bam OUTPUT= " . $FastqFile[$i] . "clean.bam"; #FILTERS UNMAPPED READS
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_remapped.bam"; #REMOVES BAM FILE
system($cmd);
```

```
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "SortSam.jar SORT_ORDER=coordinate
INPUT=" . $FastqFile[$i] . "clean.bam OUTPUT=" . $FastqFile[$i] . "sort.bam"; #SORTS
BAM FILE ON COORDINATES

system($cmd);

$cmd = "rm " . $FastqFile[$i] . "clean.bam"; #REMOVES BAM FILE

system($cmd);

$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "MarkDuplicates.jar INPUT=" .
$FastqFile[$i] . "sort.bam OUTPUT=" . $FastqFile[$i] . "dups.bam METRICS_FILE=" .
$FastqFile[$i] . "dups.metrics"; #IDENTIFIES DUPLICATE READS

system($cmd);

$cmd = "rm " . $FastqFile[$i] . "sort.bam"; #REMOVES BAM FILE

system($cmd);

$cmd = " java -Xmx" . $mem . "g -jar " . $picard . "AddOrReplaceReadGroups.jar
RGLB=Lane1 RGPL=Illumina RGPU=TTAGGC RGSM=" . $FastqFile[$i] . " INPUT=" .
$FastqFile[$i] . "dups.bam OUTPUT=" . $FastqFile[$i] . "header.bam"; #FORMATS HEADER
FOR GATK

system($cmd);

$cmd = "rm " . $FastqFile[$i] . "dups.bam"; #REMOVES BAM FILE

system($cmd);

$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "BuildBamIndex.jar INPUT=" .
$FastqFile[$i] . "header.bam"; #INDEX BAM FILE FOR GATK INPUT

system($cmd);

$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T
RealignerTargetCreator -R " . $reference . ".fasta -I " . $FastqFile[$i] . "header.bam -o " .
$FastqFile[$i] . ".intervals"; #IDENTIFIES INTERVAL TO BE REALIGNED AROUND
INDELS

system($cmd);

$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T IndelRealigner -R
" . $reference . ".fasta -targetIntervals " . $FastqFile[$i] . ".intervals -I " . $FastqFile[$i] .
"header.bam -o " . $FastqFile[$i] . "realign.bam"; #REALIGNS INTERVALS IDENTIFIED IN
PREVIOUS STEP

system($cmd);

$cmd = "rm " . $FastqFile[$i] . "header.bam"; #REMOVES BAM FILE

system($cmd);

$cmd = "rm " . $FastqFile[$i] . ".intervals"; #REMOVES INTERVALS FILE. COMMENT
THIS LINE OUT IF YOU WANT TO RETAIN THE INTERVALS FILE

system($cmd);
```

```
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T UnifiedGenotyper -
R " . $reference . ".fasta -mbq 10 -stand_call_conf 31 -stand_emit_conf 31 -ploidy 1 -I " .
$FastqFile[$i] . "realign.bam -o " . $FastqFile[$i] . "_round1_SNPs.vcf"; #CALLS SNPS
USING THE UNIFIED GENOTYPER, FILTERING ON A MINIMUM QUALITY VALUE OF
31
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_round1_SNPs.vcf.idx"; #REMOVES SNP VCF FILE INDEX
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T UnifiedGenotyper -
R " . $reference . ".fasta -mbq 10 -stand_call_conf 31 -stand_emit_conf 31 -ploidy 1 -
minIndelFrac 0.51 -minIndelCnt 3 -glm INDEL -I " . $FastqFile[$i] . "realign.bam -o " .
$FastqFile[$i] . "_INDELS.vcf"; #CALLS INDELS USING THE UNIFIED GENOTYPER,
REQUIRING A MINIMUM OF 3 READS CONTAINING THE INDEL, QUALITY VALUE
OF 31 OR HIGHER, AND A MAJORITY OF THE READS CONTAINING (0.51) THE
INDEL
system($cmd);
}

$cmd = "perl VCF_filter.pl"; #FILTERS ALL SNP VCF FILES FOR SNPS WITH <75% OF
READS AT A GIVEN SITE MATCHING THE ALTERNATE BASE CALL
system($cmd);

for ($g = 0; $g < @FastqFile; $g++){
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T
FastaAlternateReferenceMaker -R " . $reference . ".fasta -V " . $FastqFile[$g] .
"_SNPs_filtered.vcf -o " . $FastqFile[$g] . "_SNPs_reference.fasta";
system($cmd); #GENERATES NEW REFERENCE UPDATED WITH SNP CALLS
$cmd = "samtools faidx " . $FastqFile[$g] . "_SNPs_reference.fasta"; #INDEXES THE NEW
REFERENCE
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "CreateSequenceDictionary.jar
REFERENCE=" . $FastqFile[$g] . "_SNPs_reference.fasta OUTPUT=" . $FastqFile[$g] .
"_SNPs_reference.dict"; #CREATES A SEQUENCE DICTIONARY FOR THE NEW
REFERENCE
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T
FastaAlternateReferenceMaker -R " . $FastqFile[$g] . "_SNPs_reference.fasta -V " .
$FastqFile[$g] . "_INDELS.vcf -o " . $FastqFile[$g] . "_reference.fasta"; #GENERATES NEW
REFERENCE UPDATED WITH BOTH INDELS AND SNPS
system($cmd);
}
```

SECOND ROUND:

#EXPLANATIONS ARE ONLY GIVEN FOR COMMANDS THAT DIFFER FROM THE FIRST ROUND

```
#!/usr/bin/perl -w
use strict;

#assumes the bwa and samtools executables have been copied to your system path
#For this second round, each set of fastq read files has its own unique reference genome (produced automatically from the round1_assembly.pl script), and this reference genome must be in the same folder as this script and all the files.
#This pipeline assumes all barcodes have been trimmed off of the fastq reads. If barcodes are still present, simply add the '-B 6' (the 6 should be the length of your barcodes) option to the bwa aln command (lines 34 and 36).

my $i = 0;
my $cmd = "";

my $mem = 8; #memory (in gigabytes) allocated to GATK and Picard

my $stampy = '/Users/justin/Desktop/stampy-1.0.21/'; #path to stampy.py script

my $gatk = '/Users/justin/Desktop/GenomeAnalysisTK-2.1-13-g1706365/'; #path to GenomeAnalysisTK.jar

my $picard = '/Users/justin/Desktop/picard-tools-1.79/picard-tools-1.79/'; #path to all Picard java modules

my @FastqFile = ('SRR306630'); #array of input file names; this script is designed to handle only paired end read files from a single lane downloaded from the SRA. The sample file name (i.e., SRR306632) would then be followed by "_1.fastq" or "_2.fastq".

#Assembly commands

for ($i = 0; $i < @FastqFile; $i++){
```

```
$cmd = "bwa index -a bwtsw " . $FastqFile[$i] . "_reference.fasta";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "CreateSequenceDictionary.jar
REFERENCE=" . $FastqFile[$i] . "_reference.fasta OUTPUT=" . $FastqFile[$i] .
"_reference.dict";
system($cmd);
$cmd = "samtools faidx " . $FastqFile[$i] . "_reference.fasta";
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -G " . $FastqFile[$i] . "_reference " .
$FastqFile[$i] . "_reference.fasta";
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -g " . $FastqFile[$i] . "_reference -H " .
$FastqFile[$i] . "_reference";
system($cmd);
#ALL OF THE ABOVE COMMANDS INDEX THE NEW REFERENCE FOR THE
DOWNSTREAM PROGRAMS

$cmd = "bwa aln " . $FastqFile[$i] . "_reference.fasta " . $FastqFile[$i] . "_1.fastq > " .
$FastqFile[$i] . "_1.sai";
system($cmd);
$cmd = "bwa aln " . $FastqFile[$i] . "_reference.fasta " . $FastqFile[$i] . "_2.fastq > " .
$FastqFile[$i] . "_2.sai";
system($cmd);
$cmd = "bwa sampe -P " . $FastqFile[$i] . "_reference.fasta " . $FastqFile[$i] . "_1.sai " .
$FastqFile[$i] . "_2.sai " . $FastqFile[$i] . "_1.fastq " . $FastqFile[$i] . "_2.fastq > " .
$FastqFile[$i] . ".sam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_1.sai";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_2.sai";
system($cmd);
$cmd = "samtools view -bS " . $FastqFile[$i] . ".sam > " . $FastqFile[$i] . ".bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . ".sam";
system($cmd);
$cmd = "python2.6 " . $stampy . "stampy.py -g " . $FastqFile[$i] . "_reference -h " .
$FastqFile[$i] . "_reference --bamkeepgoodreads -M " . $FastqFile[$i] . ".bam -o " .
$FastqFile[$i] . "_remapped.sam";
system($cmd);
```

```
$cmd = "rm " . $FastqFile[$i] . ".bam";
system($cmd);
$cmd = "samtools view -bS -q 20 " . $FastqFile[$i] . "_remapped.sam > " . $FastqFile[$i] .
"_remapped.bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_remapped.sam";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "CleanSam.jar INPUT=" . $FastqFile[$i] .
"_remapped.bam OUTPUT= " . $FastqFile[$i] . "clean.bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "_remapped.bam";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "SortSam.jar SORT_ORDER=coordinate
INPUT=" . $FastqFile[$i] . "clean.bam OUTPUT=" . $FastqFile[$i] . "sort.bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "clean.bam";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "MarkDuplicates.jar INPUT=" .
$FastqFile[$i] . "sort.bam OUTPUT=" . $FastqFile[$i] . "dups.bam METRICS_FILE=" .
$FastqFile[$i] . "dups.metrics";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "sort.bam";
system($cmd);
$cmd = " java -Xmx" . $mem . "g -jar " . $picard . "AddOrReplaceReadGroups.jar
RGLB=Lane1 RGPL=Illumina RGPU=TTAGGC RGSM=" . $FastqFile[$i] . " INPUT=" .
$FastqFile[$i] . "dups.bam OUTPUT=" . $FastqFile[$i] . "header.bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "dups.bam";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $picard . "BuildBamIndex.jar INPUT=" .
$FastqFile[$i] . "header.bam";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T
RealignerTargetCreator -R " . $FastqFile[$i] . "_reference.fasta -I " . $FastqFile[$i] .
"header.bam -o " . $FastqFile[$i] . ".intervals";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T IndelRealigner -R
" . $FastqFile[$i] . "_reference.fasta -targetIntervals " . $FastqFile[$i] . ".intervals -I " .
$FastqFile[$i] . "header.bam -o " . $FastqFile[$i] . "realign2.bam";
```

```
system($cmd);
$cmd = "rm " . $FastqFile[$i] . "header.bam";
system($cmd);
$cmd = "rm " . $FastqFile[$i] . ".intervals";
system($cmd);
$cmd = "java -Xmx" . $mem . "g -jar " . $gatk . "GenomeAnalysisTK.jar -T UnifiedGenotyper -R " . $FastqFile[$i] . "_reference.fasta -mbq 10 -stand_call_conf 31 -stand_emit_conf 31 -ploidy 1 -out_mode EMIT_ALL_SITES -I " . $FastqFile[$i] . "realign2.bam -o " . $FastqFile[$i] . "_SNPs.vcf"; #GENERATION OF ALL-SITES VCF. QUALITY VALUE CUTOFFS ARE IGNORED IN THIS COMMAND LINE BECAUSE ALL SITES WITH AT LEAST A SINGLE READ COVERING ARE CALLED
system($cmd);
}

#Following the end of the second round of the pipeline, several additional steps are required to produce the sequence files provided in the Drosophila Genome Nexus
#POSITION ADJUSTMENT - due to aligning to an augmented reference, with INDELS called in the first round inserted into the original reference genome, positions in the ALL-SITES VCF files must be shifted back to those of the original reference genome.
#To adjust the base positions in the ALL SITES VCF files, simply place the IndelShift.pl script in the same directory as the ALL SITES VCF file(s) produced from the pipeline above, along with the round1 *INDELS.vcf files, and execute the IndelShift.pl script.
#Following position adjustment, the VCF_to_Seq scripts can be used to generate unformatted sequences for each contig in the reference genome.
#For *sites.vcf files called with the diploid (-ploidy 2) option in the Unified Genotyper, use the VCF_to_Seq_diploid.pl version, while those called with the haploid option (-ploidy 1) should use the VCF_to_Seq_haploid.pl version.
#Place the appropriate version of this script in the same directory as your *sites.vcf files. Supply the chromosome/contig names, the number of total chromosomes/contigs, the quality threshold, and the number of bases around indels to be masked (change this to zero if you want to remove this feature)
```

After generating raw sequences filtered only around indels (3-base filter), for the Drosophila Genome Nexus (DGN) we performed additional filtering of heterozygosity and identity-by-descent regions. Below, we describe these filters in detail and describe the provided scripts that can be used to perform this filtering.

HETEROZYGOSITY FILTERING

For diploid genomes (i.e., the DGRP genomes from Raleigh), we masked all heterozygous tracts. We scanned the 5 euchromatic arms of each diploid genome for heterozygous calls in 100 kb windows advancing in 5 kb increments. Rather than use a hard boundary for delineating windows of residual heterozygosity, we chose to scale the threshold for a given window to the level of genetic diversity observed in that window within either sub-Saharan or cosmopolitan populations, depending on the geographic origin of each individual genome. To determine these thresholds, we estimated nucleotide diversity (π) in 100 kb windows advancing in 5 kb increments for the large Rwandan (RG) sample of 27 haploid embryo genomes and the French sample of 9 haploid embryo genomes to represent sub-Saharan and cosmopolitan diversity, respectively. Then to scan each genome for heterozygosity, whenever the proportion of heterozygous sites in a given window exceeded π/5, a masking interval was initiated, and this interval was extended in both directions on the chromosome arm until encountering a window with heterozygosity less than π/20.

To perform this filtering, place the heterozygosity_windows.pl script in the same directory as your gzipped target *sites.vcf.gz files and execute. This will output heterozygosity in windows suitable for the heterozygosity_to_filter.pl to examine for tracts that excede the above described thresholds. To identify these tracts, the FR_pi.txt and RG_pi.txt files should be placed in the same directory as the HetRegions.txt inputs. These *_pi.txt files contain the cosmopolitan (FR_pi.txt) and African (RG_pi.txt) pi threshold windows. For cosmopolitan genomes, $thresh should be set to FR_pi.txt, whereas if the genomes are sub-Saharan African, $thresh should be set to RG_pi.txt. In addition, the objects in the @Chromosome array should be edited to match the reference contigs you wish to examine. Then, execute the heterozygosity_to_filter.pl script. Sequence ranges that excede the designated thresholds will be identified in the *HetFilterTracts.txt files that are produced.

PSEUDOHETEROZYGOSITY FILTERING

While haploid embryo genomes are not expected to contain any true heterozygosity, repetitive and/or duplicated regions can cause mismapping that results in tracts of “pseudoheterozygosity”. To detect these tracts and remove them, we implemented the same threshold approach as outlined above (without normalization, since none of these genomes showed elevated background levels of putative heterozygosity). For these genomes, the Unified Genotyper was run in haploid mode, and so read proportions were analyzed in place of called heterozygous sites. For windows with the proportion of sites with <75% of the reads matching the consensus base above π/5, that window was enucleated, and this window was extended in both directions until encountering a window below π/20.

To perform the pseudoheterozygosity filtering, the approach is the same as outlined above for true heterozygosity, with the exception of using haploid *sites.vcf.gz files as inputs, and utilizing the pseudoheterozygosity_windows.pl script to generate "heterozygosity" windows.

IDENTITY-BY-DESCENT FILTERING

Tracts of IBD may reflect the sampling of related individuals, and can contradict theoretical assumptions and complicate many population genetic analyses. To identify tracts of IBD, we implemented the approach of Pool et al. (2012), but with slight modifications for the diploid genomes and for the large DPGP3 population sample (described below). All possible pairwise comparisons were made for each of the five euchromatic arms of each genome, and pairwise differences per site were calculated in 500 kb windows advanced in 100 kb increments.

Windows with less than 0.0005 pairwise differences per site were deemed putatively IBD. Some chromosomal intervals (including centromere- and telomere-proximal regions) exhibited large-scale, recurrent IBD between populations suggesting explanations other than close relatedness, and therefore did not contribute to a genome's IBD total unless they extended outside these recurrent IBD regions. Elsewhere, within population IBD (presumably due to very recent common ancestry) was determined to be that which totaled genome-wide >5 Mb for a pairwise comparison of genomes.

For the DPGP2 and AGES genomes, we excluded the same recurrent IBD regions as those of Pool et al. (2012). However, for the much larger DGRP and DPGP3 samples of genomes, we visually reexamined these recurrent IBD tracts and generated new regions to be excluded for each of these data sets (provided in Table S4 of the DGN manuscript and available at http://johnpool.net/genomes.html).

Due to heterozygosity filtering, some diploid genomes had genomic coverages far less than the typical ~111 Mb. Therefore, the genome-wide threshold for IBD filtering was adjusted to 5% of all called positions rather than 5 Mb. In addition, only 500 kb windows with >100 kb pairwise comparisons were allowed to contribute to the 5% total, minimizing the influence of windows with large numbers of masked sites.

Pool, J. E., R. B. Corbett-Detig, R. P. Sugino, K. A. Stevens, C. M. Cardeno et al., 2012 Population genomics of Sub-Saharan Drosophila melanogaster: African diversity and