**Supplemental File 1.** ChIP-seq Computational Methods

The computations below use the TMAP sequence aligner (https://github.com/iontorrent/TMAP), the SAMtools program suite (http://samtools.sourceforge.net), the BEDTools program suite (https://github.com/arq5x/bedtools2), and the R statistical computing environment (https://www.r-project.org). The Affymetrix Integrated Genome Browser (IGB) (http://bioviz.org/igb/index.html) is used to examine the output, and for downstream data analysis using the graph thresholding and graph arithmetic tools.

TMAP, SAMtools, BEDTools are run from a unix/linux terminal window.

R can be run from a terminal window, but is usually run from its own command line environment.

The Ion Torrent Proton server processes the raw sequencing data using Torrent Suite software (https://github.com/iontorrent/TS). Although TS includes the TMAP aligner, and can be run as part of the sequence generation pipeline, we use the fileExporter plugin to generate unaligned fastq files to facilitate control over alignment, using a locally installed version of TMAP.

Steps in calculating ChIP enrichment ("$" is the terminal prompt):

1. Align the reads:

```
$ tmap mapall -f/genome/dmel5.fasta -r/data/input.fastq -g 3 -a 1 -o 2 -s/data/input.bam stage1 map4

$ tmap mapall -f/genome/dmel5.fasta -r/data/chip.fastq -g 3 -a 1 -o 2 -s/data/chip.bam stage1 map4
```

The map4 algorithm is optimized for reads longer than 100 bases.
-f gives the path to the genome fasta file
-r gives the path to the fastq sequence file
-s gives the path to the bam aligned output file
-g 3 prevents soft-clipping of reads to make an alignment.
-a 1 makes the aligner assign a read that matches more than one position in the genome to the position with the highest alignment score. This is the default and does not need to be included in the command line. If there are multiple sites in the genome with the highest score, one is chosen randomly.
-o 2 makes the aligner output an unsorted bam file.

2. Sort and index the bam files:

```
$ samtools sort input.bam input.sorted

$ samtools sort chip.bam chip.sorted

$ samtools index input.sorted.bam

$ samtools index chip.sorted.bam

$ samtools idxstats input.sorted.bam > input.sorted.stats.txt

$ samtools idxstats chip.sorted.bam > chip.sorted.stats.txt
```

These commands generate sorted bam files named "input.sorted.bam" and "chip.sorted.bam", and index files named "input.sorted.bam.bai" and "chip.sorted.bam.bai".
The index files are needed to visualize the bam files in a genome browser, which can be helpful in examining the alignment and where reads are being assigned.
The "input.sorted.stats.txt" and "chip.sorted.stats.txt" files are tabbed text files that show how many reads are assigned to each chromosome, and can be used to calculate approximate genome coverage.
Typically more than 90% of reads align.

3. Generate genome base pair coverage files:

```
$ genomecoveragebed -ibam input.sorted.bam -d > input.coverage.sgr
$ genomecoveragebed -ibam chip.sorted.bam -d > chip.coverage.sgr
```

These BEDTools commands generate text files giving the number of times each base pair is present in an aligned read. For the modified Drosophila genome, this generates a tabbed text graph (sgr) file viewable in a genome browser that is nearly 130 million lines long. The first column gives the chromosome arm, the second column gives the nucleotide position, and the third column gives the number of times that base pair is present in an aligned read. For significantly larger genomes, the length of the file will exceed the maximum 64 bit integer, and these would need to segmented for processing using this method.

4. Calculate normalized ChIP enrichment:

The following steps are performed in the R computing environment with the command line prompt “>”.

a. Load the genome coverage files and split into individual chromosomes:

```
> input <- read.table(file=“input.coverage.sgr”)

> chip <- read.table(file=“chip.coverage.sgr”)

> input.split <- split(input, input$V1)

> chip.split <- split(chip, chip$V1)
```

b. Calculate the chip enrichment for chromosome arms of interest:

```
> source(file=”chip.session.r”)
```

We use a “chip.session.r” file that can be called from the R command line to process only those chromosome arms of interest. This is useful for testing parameters such as window size, where only one chromosome is tested. The “chip.session.r” script calls the “chip.enrich.r” file that calculates enrichment and writes files for each chromosome arm, including an sgr file with the enrichment for each window, an sgr file with the log2 enrichment, and a file with statistical parameters which can be helpful in deciding what is significant enrichment. We then combine the sgr files for the various chromosome arms using the unix *cat* command (e.g. $ cat chr1.enrichment.sgr chr2.enrichment.sgr ... > chip.enrichment.sgr) for viewing in a genome browser and further data analysis. The “chip.session.r” file can be modified as needed in a text editor.

Example “chip.session.r” script file and the annotated “chip.enrich.sgr” file:

File “chip.session.r”:

```
# this uses the input.split and chip.split data arrays generated above in step 4a

    input.coverage <- input.split$"2L"
    chip.coverage <- chip.split$"2L"
    source("chip.enrich.r")

    input.coverage <- input.split$"2LH”
    chip.coverage <- chip.split$"2LH”
    source("chip.enrich.r")

    input.coverage <- input.split$"2R”
    chip.coverage <- chip.split$"2R”
    source("chip.enrich.r")

    input.coverage <- input.split$"2RH”
    chip.coverage <- chip.split$"2RH”
    source("chip.enrich.r")

# etc. for all chromosome arms to be processed
```

File “chip.enrich.r”:

```
    # this uses the input.coverage and chip.coverage data arrays generated by the “chip.session.r“ script

    # define window size and step size between windows, adjust as needed

        step.size <- 50
        window.size <- 250

    # extract array of base pair coverage values from input and chip coverage

        input.coverage.value <- input.coverage[ ,3]
        chip.coverage.value <- chip.coverage[ ,3]

    # calculate windows and steps to cover the chromosome arm and window positions

        length.coverage <- dim(input.coverage)[1] - window.size
        window.step <- window.size - 1
        midwindow.step <- (window.size %/% 2) - 1
        steps <- (length.coverage %/% step.size)

    # define data arrays to hold window positions and window coverage values

        step.position <- rep(NA, steps)
        input.step.value <- rep(NA, steps)
        chip.step.value <- rep(NA, steps)

    # initialize counters for loop that calculates window coverage values for input and chip

        j <- 1
        k <- j + window.step
        bin <- 0

    # loop to sum coverage in windows and count input windows with non-zero values

        for(i in 1:steps){
            if(sum(input.coverage.value[j:k]) > 0){
                    bin <- bin + 1
                    step.position[bin] <- j + midwindow.step
                    input.step.value[bin] <- sum(input.coverage.value[j:k])
                    chip.step.value[bin] <- sum(chip.coverage.value[j:k])
            }
            j <- j + step.size
            k <- j + window.step
        }

    # define data arrays containing only windows with non-zero values in the input

        input.bin.value <- input.step.value[1:bin]
        chip.bin.value <- chip.step.value[1:bin]
        bin.position <- step.position[1:bin]
        bin.arm <- input.coverage[1:bin, 1]

    # calculate factors to normalize for sequencing depth

        input.factor <- sum(as.numeric(input.bin.value)) / bin
        chip.factor <- sum(as.numeric(chip.bin.value)) / bin

    # add normalization factors to all window values as offsets to remove zero values in chip

        input.bin.value.offset <- input.bin.value + input.factor
        chip.bin.value.offset <- chip.bin.value + chip.factor
```

```
# calculate proportion of total coverage in each window to normalize to sequencing depth

    prop.input.bin.value.offset <- input.bin.value.offset / input.factor
    prop.chip.bin.value.offset <- chip.bin.value.offset / chip.factor

# convert proportional window values to log2 values

    log.prop.input.bin.value.offset <- log2(prop.input.bin.value.offset)
    log.prop.chip.bin.value.offset <- log2(prop.chip.bin.value.offset)

# adjust baseline values of log2 input and log2 chip proportional coverage values to 0
# corrects for under-representation of non-enriched regions in the chip sample coverage
# without correction most unoccupied regions will have negative enrichment (enrichment <1, log2 enrichment <0)
# effect is small for cohesin and Nipped-B but significant for factors with high enrichment
# under-compensates, but accuracy is sufficient for practical purposes

    log.prop.input.bin.value.offset.adj <- log.prop.input.bin.value.offset + (1 - median(log.prop.input.bin.value.offset))
    log.prop.chip.bin.value.offset.adj <- log.prop.chip.bin.value.offset + (1 - median(log.prop.chip.bin.value.offset))

# calculate the ratio of chip coverage to input coverage for all windows (fold-enrichment)

    log.enrichment <- log.prop.chip.bin.value.offset.adj - log.prop.input.bin.value.offset.adj
    enrichment <- 2 ^ log.enrichment

# calculate statistics to aid downstream analysis

    mean.log <- mean(log.enrichment)
    sd.log <- sd(log.enrichment)
    mean.enrich <- mean(enrichment)
    sd.enrich <- sd(enrichment)
    stats <- c(mean.log, sd.log, mean.enrich, sd.enrich)

# define data frames for writing output files

    log.enrichment.sgr <- data.frame(bin.arm, bin.position, log.enrichment)
    enrichment.sgr <- data.frame(bin.arm, bin.position, enrichment)

# name and write output files in sgr format (tabbed text graph files viewable in genome browser)

    fname <- paste(bin.arm[1], ".log.enrichment.sgr", sep="")
    write.table(log.enrichment.sgr, file=fname, quote=FALSE, sep="\t", row.names=FALSE, col.names=FALSE)

    fname <- paste(bin.arm[1], ".enrichment.sgr", sep="")
    write.table(enrichment.sgr, file=fname, quote=FALSE, sep="\t", row.names=FALSE, col.names=FALSE)

    fname <- paste(bin.arm[1], ".stats.txt", sep="")
    write(stats, file=fname)
```