Basic Analysis of NimbleGen ChIP-on-chip Data using Bioconductor/R (PROT43)

Tobias Straub
Adolf Butenandt Institute
Molecular Biology
Ludwig-Maximilians University, Germany

Last reviewed: 06 Apr 2009 by Fabio Mohn and Tim Roloff, Laboratory of Dirk Schübeler, Friedrich Miescher Institute, Basel

---

Introduction

Hybridization of chromatin immuno-precipitation (ChIP) material to tiling arrays at NimbleGen service facilities usually leaves the customer with a set of data files that are of limited use. Most information about the experiment is gained by either displaying immuno-precipitate(IP)/input ratio tracks (the GFF files provided) of individual hybridisation experiments with NimbleGen?s SignalMap software or by scanning the list of peaks identified by the automated data analysis. Summary profiles from replicate experiments cannot be investigated; further calculations are left to the customers. Apart from the fact that data quality cannot be directly evaluated, raw data is not corrected for systematic signal distortions in the generation of ratio GFF files. Furthermore, the robustness of the provided peak finding procedures is questionable, as the algorithm will frequently identify many "significant" peaks in noise-only experiments.

Several independent software tools are available that provide a more robust and more detailed analysis. Unfortunately, those tools frequently tend to underperform on datasets that are largely different from the ones they have been built for. Apart from simple problems such as compatibility issues between different organisms and naming conventions for chromosomes many tools aim, e.g., for identifying peaks or peak centers. In the realm of epigenetics, however, many features are distributed rather broadly and peak centers are not the only features defining biological states. Overall, many tools do not provide sufficient flexibility and transparency for the user to know and control what is actually happening with the data.

This will ultimately lead to non-reproducibility and/or analysis failures once the tools are modified.

With increasing amounts of quantitative data biologists are often left with endpoints of analyses that they simply have to trust if they are not given the possibility to look behind the procedures. This protocol mainly aims for aiding the biologists to get a rather unbiased look at the quantitative data of their NimbleGen tiling array experiments during initial processing steps. Ultimately, summary tracks of the profiles will be generated that permit visual browsing of the data, which serves as important inspiration for downstream analyses. The procedures introduced here can in principle applied to any other 2-color tiling array data (Agilent) and/or 2-sample comparison data on Affymetrix chips (single colour).

Bioconductor/R

R is a free and platform-independent scripting framework for statistical analyses (Click here for more information). Its modular structure allows for integration of specific tools such as the Bioconductor modules for bioinformatic applications (Click here for more information). These extensions facilitate processing of biological data by providing easy-to-use interfaces to statistical core modules and by adding specific algorithms such as normalization steps for microarray analyses. Other highlights of R/Bioconductor are its excellent graphing capabilities, the scriptability which allows for automatization of complex analysis pipelines, and the existence of a large community creating and maintaining state-of-the-art tools. Most importantly, however, data in R is ready for explorative examination (for example correlations) and complex downstream analyses.

Unfortunately, data handling in R is driven by commands typed into a terminal window. This implies a rather steep learning curve, but there are several reasons not to shy away: At first there is plenty of documentation on R usage for biologists. I recommend the public domain resources "R-introduction for Biostatistics" by Kim Seefeld (Get Pdf version) and the "R & BioConductor Manual" by Thomas Girke (View HTML version). In addition, there are several good introductory books published on R/Bioconductor ("Bioinformatics and Computational Biology Solutions Using R and Bioconductor" by Robert Gentleman et al. and "Bioconductor Case Studies (Use R!)" by Florian Hahne et al.). Secondly, the commands to be typed can simply be copy-pasted or loaded from preformed text files. The analysis steps executed in this protocol can therefore be adjusted to any other NimbleGen dataset by editing a few lines in a sample description file. This protocol does not require the reader to be familiar with R/Bioconductor.

Rationale of the analysis pipeline

Quality control: the validity and robustness of any analysis will depend on the quality of the raw data. The most important quality parameters are the distribution of raw signal intensities and the independence of IP/input ratio and absolute signal strength. In addition, arrays with severe hybridisation artefacts should be identified.

Normalization: shifts in signal intensity distributions between the two channels as well as between arrays have to be adjusted. Dependences of ratio and absolute signal strengths should be eliminated.

Probe summary statistics: For each probe on the array a mean ratio value of all replicates and a statistical test for enrichment is provided.

Region summarization: The spatial organization of bound probes can be exploited to identify statistically significant regions of binding.

Data visualization: The results of the analyses will be exported for visualization in genome browsing applications.

Procedure

Installation of R/Bioconductor

Get the recent version of R for your platform by choosing a mirror server next to you in http://cran.r-project.org/mirrors.html. Follow the installation instructions provided. GUI versions of R are launched by double-clicking the corresponding icons in OSX or Windows. Linux R requires issuing the command 'R' in a terminal window.

In addition to the R base packages provided with the standard installation, this protocols requires some modules, which in part are bioconductor packages. The following chain of commands pasted into the R terminal will install all packages required. The installation will take some time and require some hard disk space (1-3 GB).

Code presented in here is preceded by the default R command prompt (>), which should not be part of the pasted command. No details on what is actually performed by the commands are provided here. The entire code (also the one for installation) without the command prompts is provided (http://genome1.bio.med.uni-muenchen.de/downloads.htm) in fully commented text files (noe_installation.R) that can either be copy-pasted or executed using the source command - source("noe_installation.R") - from within the terminal window.

```
> install.packages("st", dependencies=T)
> install.packages("locfdr", dependencies=T)
> install.packages("tileHMM", dependencies=T)
> install.packages("gplots", dependencies=T)
> source("http://bioconductor.org/biocLite.R")
> biocLite("geneplotter")
> biocLite("limma")
> biocLite("vsn")
```

The example dataset

Here I demonstrate the protocol on a published data set (Straub et al, 2008). The target protein is Drosophila MSL2, a member of the Dosage Compensation Complex that specifically binds the X chromosome in male flies. The data comprises 3 biological replicates and includes one dye swap. Immuno-precipitated (IP) and Input material was hybridized to a custom NimbleGen array that uses an isothermal design layout (i.e. oligo probes of varying lengths).

Organizing the workspace folder

All data belonging to the experiment are organized into one folder (Figure 1). This comprises the corresponding *.pair files found in the RawDataFiles folder of the service DVD and the *.ndf and *.pos files found in the layout folder within the DesignInformation folder. The experiment folder should also contain a description file of the samples (sample_key.txt, see below), an R script files that defines functions for some more complicated routines (noe_source.R) and an R script file that contains the commands of the analysis pipeline (noe_protocol.R).

Figure 1: Organization of files

The fully commented R script files can be downloaded from here: http://genome1.bio.med.uni-muenchen.de/downloads.htm

An example folder containing the raw data and layout information, as well as sample_key and R-script files can be obtained here: http://genome1.bio.med.uni-muenchen.de/dl/protocol.zip

Creating and editing of the sample key file

The use of a sample description file facilitates recycling of the R-scripts in that experiment-specific data is kept outside of the script. The file 'sample_key.txt' provided here is a tab-delimited table that can be created or edited in Microsoft Excel (Figure 2). Please note that the file has to be saved as a tab-delimited text file. For this protocol 3 columns are required: a column named 'file' that contains the name of the raw data file (*.pair), a column 'sample.type' that contains 'e' for enriched if IP material has been hybridized or 'i' for input if total input chromatin has been used. In addition, the column 'sample.name' groups the corresponding raw data from the two channels of the sample array.

	A	B	C
1	file	sample.type	sample.name
2	145215_532.pair	i	145215
3	145215_635.pair	e	145215
4	145216_532.pair	e	145216
5	145216_635.pair	i	145216
6	305059_532.pair	i	305059
7	305059_635.pair	e	305059

Figure 2: Contents of the sample key file

Reading the raw data

Launch R and navigate to your experiment folder. In R.app for OSX this is simply achieved by dragging the folder icon onto the R icon in the launch bar. Otherwise set the working directory accordingly using the R command "setwd()". To check that you are in the proper working directory test the contents of the folder using "list.files()".

```
> setwd("/Path/to/your/dataset1")
> list.files()
```

Reading of both raw data and layout information is performed by calling specific functions that are provided in the file 'noe_source.R' and get activated upon sourcing the script file. At first the sample key file is read to extract information on the raw data files. The raw signal intensities are then loaded into a matrix with columns corresponding to hybridization channels

and rows corresponding to probes. The name of the row is the Probe ID as provided in the NimbleGen layout.

The layout information is extracted from the headline of the first raw signal file read. This should correspond to the name of the *.ndf and *.pos file in the folder. Parts of the layout and position information are then read into a data frame (comment 1). Non-experimental probe information is omitted from both signal and layout tables. Only experimental probes are included in the processing pipeline, therefore the number of table rows - each corresponding to one probe - is smaller than the actual number of probes on the array. Both the signal matrix and the layout frame are sorted by id of the probe. This will ensure that a probe has the same row index in both tables.

After reading of the data, the tables for sample description, raw signals, and layout are saved into separate R data files. This speeds up reading of the raw data next time they are needed.

As mentioned before, all code is available as separate text files. Whereas in here, the commands executed are not explained, the code in the text file is fully commented to give insight into what is actually happening and what all the parameters mean. Please paste code from in here line-by-line omitting the > in front of each line.

```
> source("noe_source.R")
> sample.key.file <- "sample_key.txt"
> samples <- read.delim(sample.key.file, header=T,
comment.char="#", stringsAsFactors=F)
> samples
>
> signals <- read.pairs(samples$file,
samples$sample.id)
> head(signals)
> layout.name <- get.layout.info(samples$file[1])
> layout <- read.layout(layout.name)
> head(layout)
>
> save(samples, file="samples.rda")
> save(signals, file="signals.rda")
> save(layout, file="layout.rda")
```

Perform quality controls!

At first we look at the distribution of signal intensities of the single channels (Figure 3). In an ideal case, the log-transformed signals should be close to normal distribution, which is an important prerequisite for a good signal-to-noise ratio.

Subsets of the signal matrix can be generated by calling signals[i,j] where i defines row(s) and j column(s). signals[,1] will therefore extract the raw intensities of the first raw data file read. Changing j to 2 to 6 will address the other channels and hybridisations accordingly. The Q-Q

plots will allow a better estimation of normality than simple density plots. Here we even can look at the perfect normality indicated by the red dashed line created by qqline().

```
> plot(density(signals[,1]))
>
> par(mfrow=c(1,2))
> plot(density(log2(signals[,1])))
> qqnorm(log2(signals[,1]), pch=".")
> qqline(log2(signals[,1]), lty=2, col=1)
```

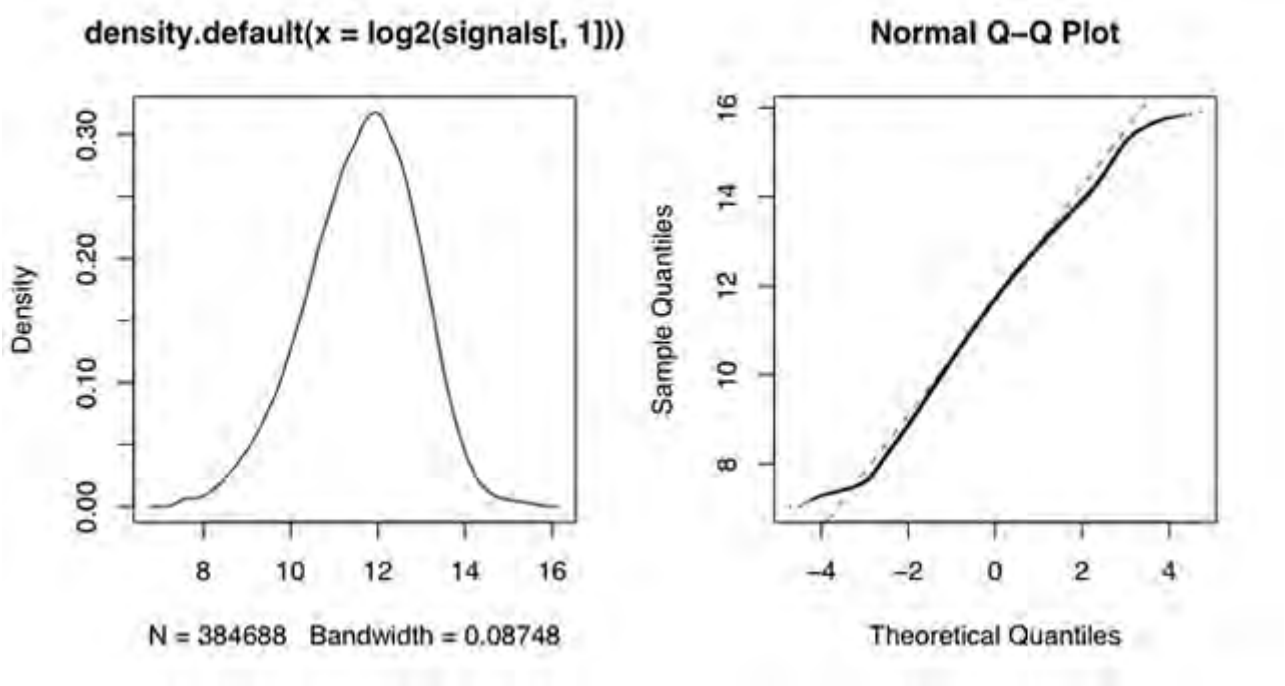

Figure 3: Distribution of raw signal intensities (density plot and Q-Q plot) of channel one.

The plots shown here indicate an almost perfect distribution of the raw signal intensities. You might want to have a look at the hybridisation of the array with sample name '305059', signals[,6]. Here, a saturation effect can be observed.

Surrogate for factor binding or enrichment is the IP/input ratio of our signals. It is therefore essential that the ratio is independent of the absolute signal strength. Very often in microarray studies this is not the case and frequently ratios increase with increasing signals. The systematic deviation has to be corrected for by normalization measures. We display this dependence using MA plots where M indicates the ratio and A the average signal (Figure 4). A similar problem regards the variance or standard deviation of the signals. The stronger the skews are the stronger the counter measures have to be and the more the raw data will be manipulated leading almost certainly to higher noise than obtained for raw data of better quality (comment 2). Eventually, noisy profiles will almost certainly increase the number of false positive binding events.

```
> par(mfrow=c(1,2))
> A = (log2(signals[,2]) + log2(signals[,1]))/2
> M = (log2(signals[,2]) - log2(signals[,1]))
> require(affy)
> require(vsn)
> ma.plot(A, M, plot.method="smoothScatter",
cex=0.8, main="MA plot")
> meanSdPlot(log2(signals[,1:2]), ranks=TRUE,
main="SD versus rank")
```

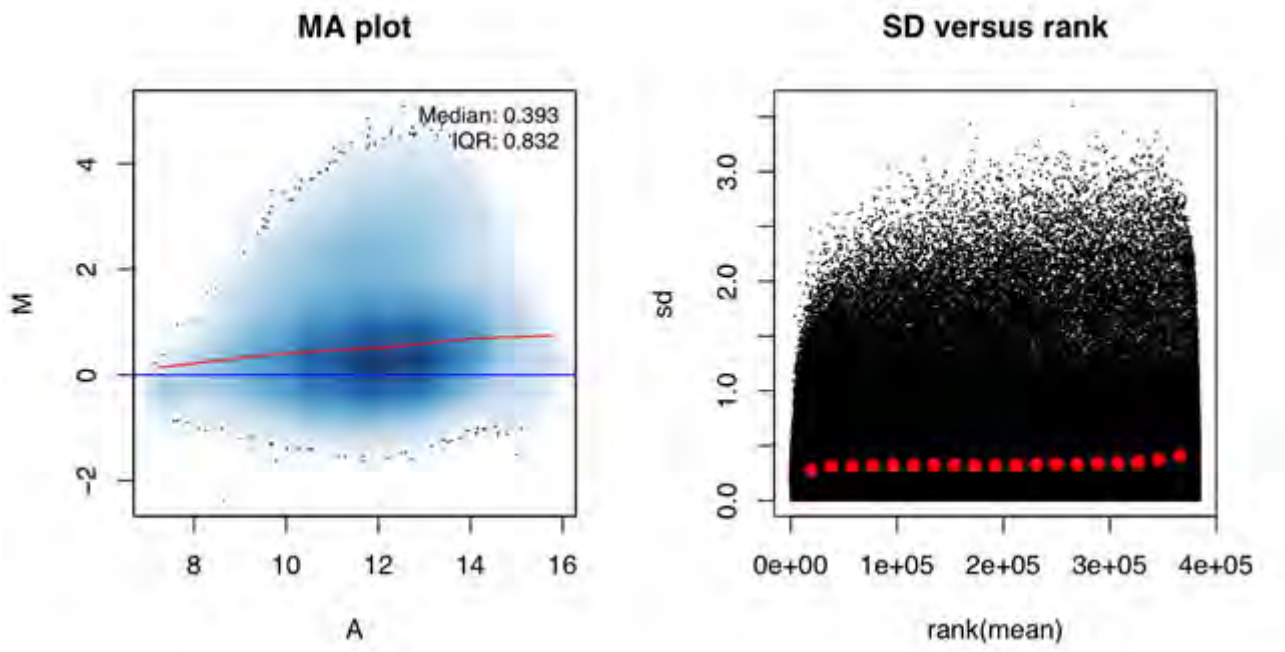

Figure 4: Plots for visualization of ratio (MA plot) and variance (SD versus rank) dependence on absolute signal intensities.

The un-normalized signals for array #1 do not indicate major skews and we can happily include the data in our experimental set.

A page combining all diagnostic plots above for one array (indicated by its sample name in the samples table) with 2 channels can be obtained using a predefined function:

```
> quality.control(signals, samples, "145215")
```

Next, we want to have a look at the reconstructed array image in order to identify major artefacts that occurred during the processing steps such as scratches on the array surface. Information on the physical position of each probe is given in the X and Y columns of the layout table. This code will display the pseudo-coloured image of one channel (Figure 5).

```
> nimble.image(log2(signals[,1]),layout)
```

As the probes are organized randomly across the arrays, localized signal distortions are most likely due to technical artefacts. In this channel - and also in the corresponding second channel (signals[,2]) - one can easily identify a little scratch in the lower left quadrant. Artefacts like this are not problematic and as they usually occur in both channels they are eliminated by ratio calculation. Larger defects should be analyzed further to estimate their impact on the entire data set.

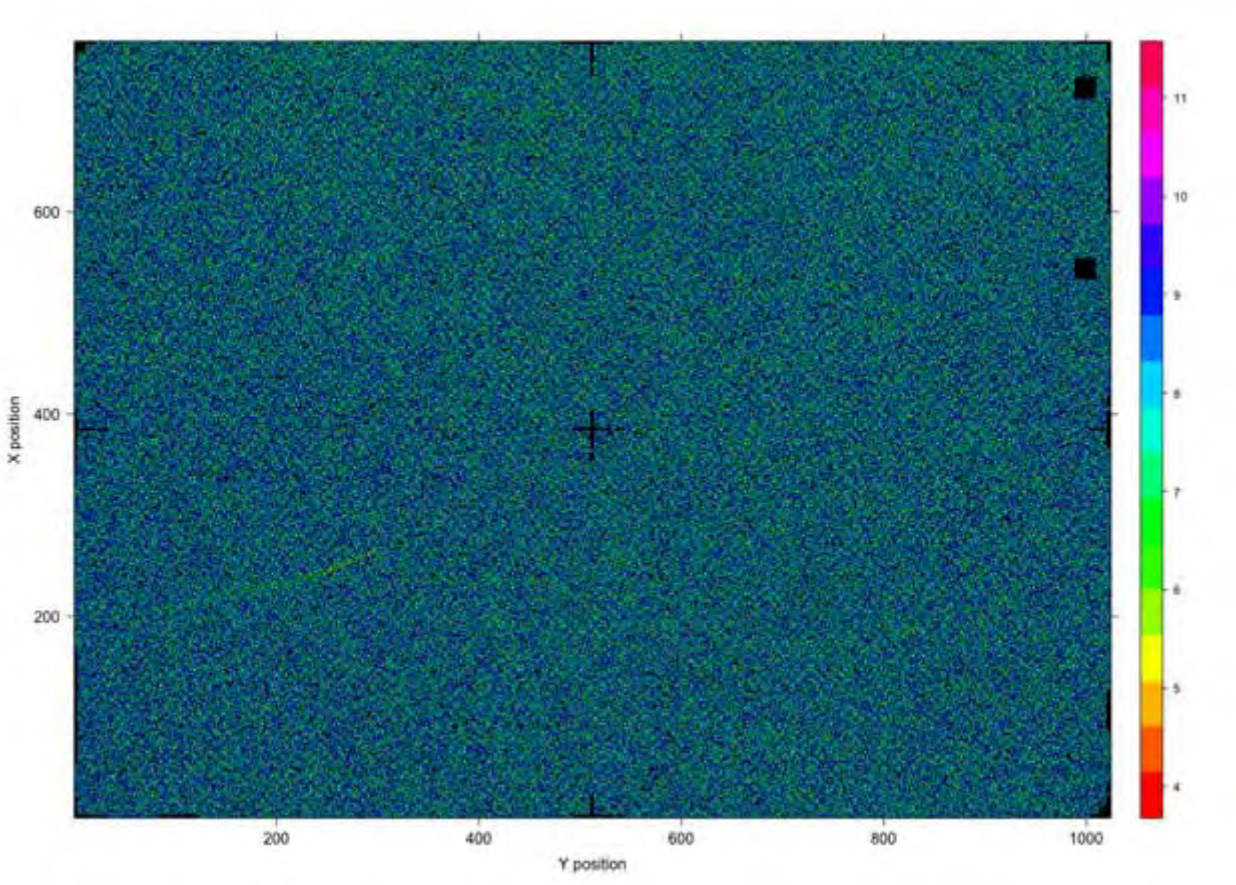

Figure 5: Reconstructed array image of channel one.

Finally, pair wise correlation plots of all signals in an experimental set might facilitate the identification of outlier hybridizations that might have to be excluded from analysis. The following code provides pair wise scatterplots of all channels as well as pair wise spearman correlation coefficients.

```
> correlate.batch(log2(signals))
```

In summary, the quality control of the data set used here allows for inclusion of all arrays in further processing steps. Mild problems can be observed for array #3 that can be visualized in signal distribution plots as well as pair wise correlation plots with the other arrays.

Normalizing the data

Inter-array comparison shows that the raw signals of our 3*2 channels are spread quite differently. Furthermore there might be some ratio and variance distortions based on signal intensities. We therefore introduce a data normalization step that should correct for most of these problems. Many different algorithms have been proposed, most of which perform equally well on high quality data. Here, I suggest the vsn normalization (Huber et al, 2002) provided by the 'vsn' package from bioconductor. This package is particularly suited to address all problems mentioned above, while other procedures like quantile normalization mainly corrects for the spread of the data (comment 3). In cases of severe distortions, other normalization procedures might be tested. However, one should keep in mind that strong distortions will lead to high noise even (and a high false positive rate of binding definition) with or even more with correction by normalization. Therefore the primary aim should always be to generate good raw data.

The vsn normalization will also log transform our raw data that we put into a new matrix named 'normalized.signals'. We will plot the signal distribution before and after normalization to clarify the effect (Figure 6).

```
> require(vsn)
> normalized.signals <- exprs(vsn2(signals,
subsample=20000, verbose=F))
>
> par(mfrow=c(1,2))
> boxplot(log2(signals)~col(signals), main="before
normalization")
>
boxplot(normalized.signals~col(normalized.signals),
main="after normalization")
```

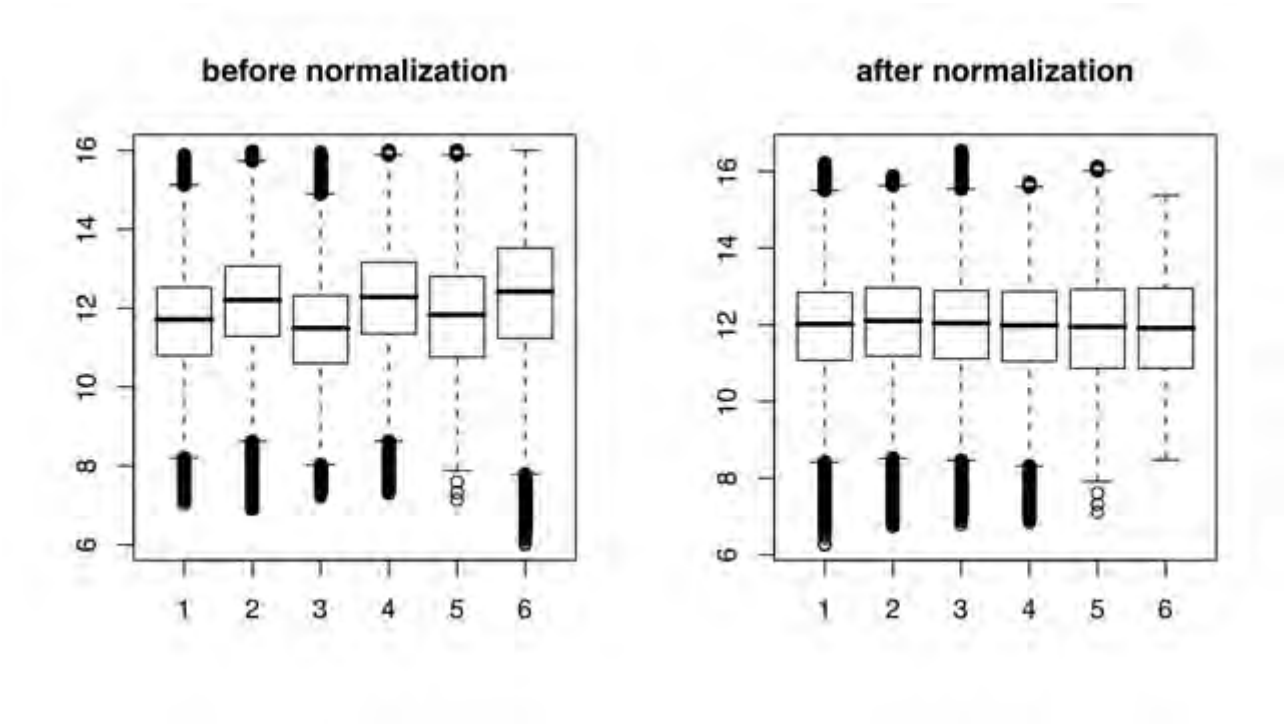

Figure 6: Signal distributions of all experimental channels before and after vsn normalization.

The effect on ratio biases can be shown with the following code. Note that a) the initial skew is very mild and b) the vsn normalization creates some small additional distortions on the data.

```
> require(affy)
> par(mfrow=c(1,2))
> A = (log2(signals[,6]) + log2(signals[,5]))/2
> M = (log2(signals[,6]) - log2(signals[,5]))
> ma.plot(A, M, plot.method="smoothScatter",
main="before normalization", cex=0.8)
> A = (normalized.signals[,6] +
normalized.signals[,5])/2
> M = (normalized.signals[,6] -
normalized.signals[,5])
> ma.plot(A, M, plot.method="smoothScatter",
main="after normalization", cex=0.8)
```

Identification of enriched probes

We now test every single probe for significant enrichment across all replicates (comment 4). This procedure involves applying a test statistics on all signals of each probe with subsequent estimation of its local false discovery rate (fdr). Fdr determination allows for the correction of the massive multiple testing problem on high-density tiling data. This problem would lead to a

very high rate of false positives if a cut-off on the test statistic alone would be used for defining bound probes.

Among the many testing algorithms available we chose 'SAM', which in our hands performs well on different array platforms and targets (Tusher et al, 2001). In principle other algorithms can easily be implemented as the 'st' package used here nicely interfaces with different tests. In addition, we also calculate the simple mean of ratios for each probe.

```
> pmean <- probe.mean(normalized.signals,
samples$sample.type)
> plot(density(pmean))
>
> pstat <- probe.stat(normalized.signals,
samples$sample.type)
> plot(density(pstat), main="sam statistic")
```

Next, we calculate the local false discovery rate based on the sam statistic. Locfdr (Efron, 2007) will display a diagnostic plot highlighting fitting lines of the total distribution as well as the (non-enriched) null distribution. It might be necessary to adjust the number of degrees of freedom if the fit is bad, but most of the cases a small misfit won't matter. After setting a cut-off at a false discovery rate of 0.02 (cut-off values up to 0.2 might be used) we then visualize the distribution of the probes in the histogram of the mean ratios (Figure 7). Obviously all probes we now defined as enriched gather in the right tail of the distribution.

```
> require(locfdr)
> par(mfrow=c(1,1))
> lfdr <- locfdr(pstat)$fdr
>
> cutoff <- lfdr < 0.02
> require(geneplotter)
> x <- list(pmean[cutoff], pmean[!cutoff])
> histStack(x, breaks=seq(-2,4,0.125), col=c(2,0))
```

Figure 7: Mean ratio histogram of statistically defined enriched and non-enriched probes.

Identification of enriched regions

Additional power for identifying bound regions is obtained by including the spatial organization of the probes. This allows for definition of enriched states not only on the enrichment ratio of the single probe but by an increased enrichment on a run of several adjacent probes. Usually this step involves either a so-called sliding-window approach or hidden markov modelling. Both methods have their variations and limitations. In downstream analyses I usually try to avoid working with categorized data (i.e. bound and unbound definitions), therefore I do not spend too much time finding the best region summarization method. Here I introduce a HMM package that is fairly easy to operate from within R and that

produces plausible results on good data (Humburg et al, 2008). The analysis will produce a vector pstate that defines for every probe whether it is in a bound region (value 1) or unbound region (value 0). Furthermore the function will return a table of regions that have been identified as enriched or bound.

```
> hmm.out <- stateHMM(stat, layout)
> pstate <- hmm.out$probe.state
> regions <- hmm.out$regions
>
>x <- list(pmean[state==1], pmean[state!=1])
> histStack(x, breaks=sea(-2,4,0.125), col=c(2,0))
```

Looking at the histogram (Figure 8) now indicates that the hmm function picks up more probes than thresholding on the fdr of the individual probe statistic. However, thresholding is in general a rather subjective measure with little biological meaning. One should not stick too much to the calculated bound and unbound categories, as there are probably many in-between states that are biologically meaningful.

Figure 8: Histogram of enriched and non-enriched probes identified by a HMM based region summarization approach.

Export the data for visualization

After we looked only at summarized data so far, the next step involves displaying the computed values along the genome. While it is possible to perform these plots within R, the export of the data and import into one of the genome browsers is more convenient. We will export continuous track information using either the GFF, SGR or WIG format depending on the genome browser. Here I chose the WIG format to export mean ratio, sam statistic and hidden markov state (comment 5). Bound regions are written into a GFF file. All files can be read and displayed in the Integrated Genome Browser (http://igb.bioviz.org/). The genome corresponding to our data here is the 2004 version of Drosophila melanogaster, which has to be loaded before data import (Figure 9).

```
> write.wiggle(pmean, layout, "mean.wig",
descr="mean")
> write.wiggle(stat, layout, "stat.wig",
descr="stat")>
> write.wiggle(state, layout, "state.wig",
descr="hmm state")
> write.gff.regions(regions, "bound_regions.gff")
```

Figure 9: Visualization of tracks generated in R/Bioconductor using the Integrated Genome Browser. From top to bottom: mean log2 ratio (orange), sam statistic (turquise), HMM state (violet), and bound regions (pink).

Outlook

The analysis pipeline presented here should be applicable to any NimbleGen experiment. Downstream analyses such as detection of distinct binding patterns are usually very specific for the chosen target and require very specific approaches. As mentioned above, R/Bioconductor provides an excellent platform for further computations and explorative analyses.

In theory, raw data from other platforms such as Affymetrix or Agilent can be processed with the same pipeline. Furthermore, data in MIAME format deposited at public repositories (GEO or ArrayExpress) can be analyzed as well. However, due to the different formats the data has to be read into the corresponding signals and layout tables in different way. Once these tables are generated processing can be performed using the same function calls.

Reviewer Comments

Reviewed by: Fabio Mohn and Tim Roloff, Laboratory of Dirk Schübeler, Friedrich Miescher Institute, Basel

1. Due to rather frequent and not obvious layout changes in the raw data (.pair) and also in the array design files (.ndf, .pos), names and column position might have to be adjusted in the read.layout function in the ?noe_source.R? file in order to ensure correct loading of the required data. The same should be kept in mind when using other Nimblegen array designs.
2. As already pointed out in the protocol we would like to emphasize that normalization can not make poor raw data look good.
3. VSN normalization is model-based data transformation according to parameters estimated form the non-normalized data. Often two color arrays show dye specific skews which can lead to an over-correction when using VSN normalization. For inter-array normalization we prefer to use scale normalization which is a less stringent normalization contained in the limma package [normalizeBetweenArrays(data_object, method=?scale?, ?)]. In this case however, the inter-array normalization does not take care of any kind of background problems or signal intensity distortions. Thus, in case one channel of an array is skewed in the low signal intensity range additional normalization such as loess normalization [limma package normalizeWithinArrays(data_object, method=?loess?, ?)] can be performed prior to inter-array normalization.
4. The probe level statistics section only works if the arrays represent replicate experiments, thus for comparing different experimental conditions, one needs to run the analysis separately.
5. The wiggle output files are tailored towards presentation in the integrated genome browser. Slight adjustments to the formats have to be made to display the files in UCSC genome browser or similar visualization tools.

References

1. Efron B (2007) Correlation and Large-Scale Simultaneous Significance Testing. Jour Amer Stat Assoc **102**: 99-103

2. Huber W, von Heydebreck A, Sultmann H, Poustka A, Vingron M (2002) Variance stabilization applied to microarray data calibration and to the quantification of differential expression. Bioinformatics **18 Suppl 1:** S96-104
3. Humburg P, Bulger D, Stone G (2008) Parameter estimation for robust HMM analysis of ChIP-chip data. BMC Bioinformatics **9**: 343
4. Straub T, Grimaud C, Gilfillan GD, Mitterweger A, Becker PB (2008) The chromosomal high-affinity binding sites for the Drosophila dosage compensation complex. PLoS Genet **4**(12): e1000302
5. Tusher VG, Tibshirani R, Chu G (2001) Significance analysis of microarrays applied to the ionizing radiation response. Proc Natl Acad Sci U S A **98**(9): 5116-5121