Chapter 3

Two-Dimensional Data Binning for the Analysis of Genome Architecture in Filamentous Plant Pathogens and Other Eukaryotes

Diane G.O. Saunders, Joe Win, Sophien Kamoun, and Sylvain Raffaele

Abstract

Genome architecture often reflects an organism's lifestyle and can therefore provide insights into gene function, regulation, and adaptation. In several lineages of plant pathogenic fungi and oomycetes, characteristic repeat-rich and gene-sparse regions harbor pathogenicity-related genes such as effectors. In these pathogens, analysis of genome architecture has assisted the mining for novel candidate effector genes and investigations into patterns of gene regulation and evolution at the whole genome level. Here we describe a two-dimensional data binning method in R with a heatmap-style graphical output to facilitate analysis and visualization of whole genome architecture. The method is flexible, combining whole genome architecture heatmaps with scatter plots of the genomic environment of selected gene sets. This enables analysis of specific values associated with genes such as gene expression and sequence polymorphisms, according to genome architecture. This method enables the investigation of whole genome architecture and reveals local properties of genomic neighborhoods in a clear and concise manner.

Key words Genome architecture, Data binning, Intergenic, R, Visualization, Heatmap, Effectors, Filamentous plant pathogen

1 Introduction

Comparative genomic analyses have revealed that evolutionary constraints often disparately affect coding, regulatory, and non-coding sequences. In addition, the distribution of these constraints largely depends on the relative position of genetic elements in the genome, referred to as the "genome architecture" [1]. In eukaryotic genomes, the organization of genes and genomic neighborhoods has evolved to ensure accurate regulation of gene expression and splicing. However, the plasticity of their structure can exceed that of their sequence, leading to structural re-arrangements over short evolutionary scales whereas genetic sequences, which are under strong constraints, diverge only minimally. The evolution of genome architecture is largely driven by genetic drift, but is also

Paul Birch et al. (eds.), *Plant-Pathogen Interactions: Methods and Protocols*, Methods in Molecular Biology, vol. 1127, DOI 10.1007/978-1-62703-986-4_3, © Springer Science+Business Media New York 2014

constrained by lifestyles and specific biological contexts [1]. Therefore, studies of genome architecture and its evolution can provide significant insight into gene regulation and function, and the underlying molecular bases of adaptation.

The recent generation of genomic data for plant pathogenic fungi and oomycetes has revealed an expansion in genome size in several unrelated lineages [2–4]. For example, in the late blight oomycete pathogen *Phytophthora infestans*, a repeat-driven expansion of the genome has created repeat-rich, gene-sparse regions that are distinct from the gene-dense conserved regions [2]. For every gene, the distance to its closest gene neighbors in either direction, designated as its flanking intergenic regions (FIRs), can be used to determine whether a gene resides in a gene-dense or gene-sparse environment. A dramatic enrichment in genes associated with virulence is observed in *P. infestans* repeat-rich, gene-sparse regions, [2, 5]. Accordingly, most *P. infestans* effector genes have longer FIRs than the genome average [6, 7]. Although less striking than *P. infestans*, repeat-rich genomic niches harboring pathogenicity-related genes are also found in several other filamentous plant pathogens, including *Magnaporthe oryzae* telomeric regions, *Leptosphaeria maculans* AT-rich isochores, and *Fusarium* spp. conditionally dispensable chromosomes [8]. Genes encoding *Vir* antigens in the malaria parasite *Plasmodium falciparum* reside in repeat-rich regions of the genome suggesting that some animal eukaryotic parasites have also evolved repeat-rich regions that host pathogenicity-associated genes [9].

Given that genes associated with pathogenicity tend to have long FIRs in some pathogen genomes, genome architecture can, in some cases, be used to identify new candidate pathogenicity genes. For example, the oomycete pathogen *Pythium ultimum* lacks a key family of oomycete effectors, known as "RXLR-type" effectors that contain a conserved Arginine-X-Leucine-Arginine N-terminal motif involved in translocation into host cells [10, 11]. Analysis of the architecture of the *P. ultimum* genome combined with Markov clustering resulted in the identification of a novel family of the candidate effector genes encoding small-secreted proteins with a conserved N-terminal domain. This domain contains a YXSL[RK] motif strongly enriched in secreted proteins and typically located between amino acid 60 and 80, similar to the RXLR motif [10, 11]. This example clearly illustrates how examining genome architecture can help to reveal novel effector candidates.

Analysis of genome architecture through the visualization of the length and distribution of intergenic regions is one way to gain insights into the biology and evolution of eukaryotes. Eukaryotic genomes are typically comprised of between 5,000 and 50,000 genes, each flanked by two intergenic regions (5′ and 3′ FIRs). To facilitate the mathematical analysis of such large datasets,

"quantization," the process of mapping a large set of input values onto a smaller set, is often used. Data binning is a quantization method, replacing values that fall within a given interval (called a "bin") by a value representative of this interval, thereby reducing the number of values to analyze.

Here we report a method that was developed using two-dimensional data binning to visualize both the 5′ and 3′ intergenic regions on either side of each gene in a single representation. In this approach, genes are sorted into bins that are defined firstly by the length of the 5′ intergenic regions (first dimension) and secondly by the length of 3′ intergenic regions (second dimension). In the simplest implementation of the method, the representative value of a bin is the number of genes it contains. This is represented by a color code in a heatmap or as a third dimensional value in a surface plot, providing a view of gene-density distribution [2, 12]. Using this method the genome architecture for a whole genome can also be compared to a smaller dataset, such as the length of intergenic regions for a small subset of genes. Indeed, the method described here offers the opportunity to overlay a scatter plot of a particular subset of genes on the whole genome heatmap, thereby highlighting the position of selected genes in the genome architecture. This approach was used to compare the FIRs of effector genes to the architecture of whole genomes in *Phytophthora* spp., illustrating that effector genes preferentially reside in gene-sparse regions of these genomes [2, 5–7].

Data binning is also used as a pre-processing technique to accelerate analyses and reduce the bias caused by minor observation errors in metagenomics [13], mass spectrometry imaging [14], and modeling [15]. In comparative genomics, data binning has proved useful in revealing local properties of genomic neighborhoods. Values describing gene properties can be associated with every gene and processed per bin, providing a representative value for genes in the same bin that have FIRs of similar length. Using *in planta* expression data as an associated value, *P. infestans* gene-sparse regions were shown to be enriched in transcriptionally induced genes during plant infection [16]. Analyses of the frequency and the type of single nucleotide polymorphisms and gene copy number as associated values led to the concept of a "two-speed genome," with gene-sparse regions evolving faster than gene-dense regions in the *P. infestans* lineage [16].

We describe here a method to perform two-dimensional binning of genes based on the length of their FIRs, overlay a scatter plot over the resulting heatmap graph, and analyze the distribution of gene-associated values such as gene expression and sequence polymorphisms, according to genome architecture. The method, implemented in R, takes standard gff or gtf coordinate files as input and produces datasets and graphics that can be

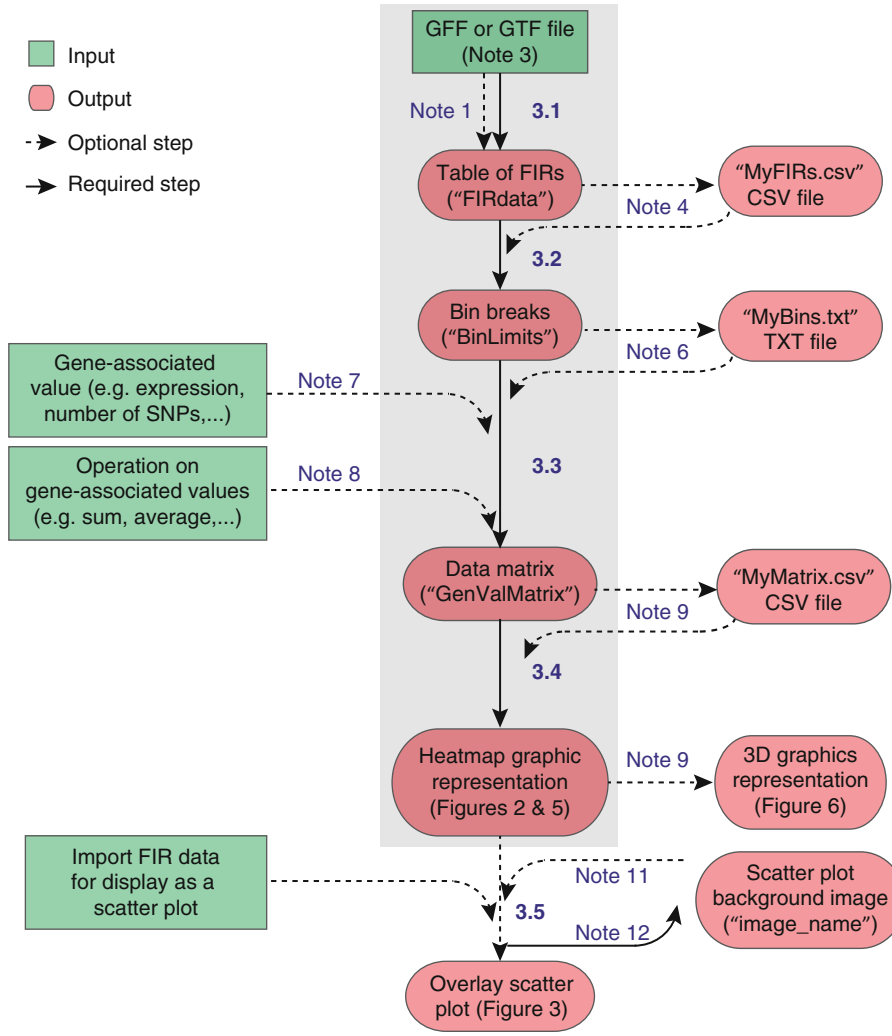

Fig. 1 Overview of the method showing required and optional analysis steps. Numbers refer to Subheadings 3 and 4. The core of the method is shown in a *gray box*, with optional inputs (*green*) and outputs (*red*) connected with *dotted arrows*

exported in any format handled by R. It is composed of five major sections (Fig. 1):

1. Calculation of FIRs length based on gff/gtf input.
2. Determination of optimal bins based on quantiles and exponential regression.
3. Data binning.
4. Representation of binned data as a heatmap graph.
5. Optional: Overlay of a scatter plot representing FIRs for a subset of genes.

Intermediary output files can be exported or imported at each major step of the analysis, as explained in the corresponding notes and outlined in Fig. 1.

The shape of genome architecture heatmaps is largely dependent on the choice of the bin size. If a bin size is too large local variations in genome environment are collapsed. Conversely, if a bin size is too small gene frequencies fluctuate greatly and patterns of genome architecture cannot be distinguished. In many eukaryotic genomes, the length of intergenic regions in a genome roughly follows a Gaussian distribution. Considering the form of Gaussian functions, we approximated that the length of intergenic regions varies proportionally to the logarithm of gene frequencies. To fit with this property, the proposed method determines bins of size increasing exponentially.

For global analysis of genome architecture, section five can be disregarded. The methods for data binning (Subheading 3.3) and **Notes 7** and **8** describe how to associate values with genes, providing the opportunity to analyze the distribution of these values according to genome architecture. In the following command lines, parameters to be set by the user are underlined. All commands should run if typed in or copied in the R console as shown, omitting the initial "> " prompt. Long command lines are split across lines for clarity reasons, and should run if typed in as a single line or copied in the R console as a whole (omitting the initial "> ").

2 Materials

2.1 Required R Packages

1. The following R base packages are required: "base," "graphics," "grDevices," "stats," "utils," installed by default with R.2.XX.X. We recommend running R 3.0.1 or above for better compatibility.
2. Three packages from the bioconductor suite are needed, and they can be downloaded and installed with the following instructions:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("GenomicRanges")
> biocLite("rtracklayer")
> biocLite("Rsamtools")
```

3. Four packages hosted by the CRAN mirror sites can be downloaded directly using the "install.packages" function as follows:

```
> install.packages("png")
> install.packages("gridExtra")
> install.packages("ggplot2")
```

2.2 Required Custom R Functions

1. The methods use the function "getFeat2" or "getFeat2b", modified versions of the "getFeat" function contributed by Thomas Girke. The codes of the "getFeat2" and "getFeat2b" functions can be downloaded from the Figshare repository using the links http://dx.doi.org/10.6084/m9.figshare.707325 and http://dx.doi.org/10.6084/m9.figshare.707326 respectively.
2. The methods use the function "filled.contour3" that can be downloaded using the link http://dx.doi.org/10.6084/m9.figshare.707327 to the Figshare repository. This is a modified version of the "filled.contour" function contributed by Ian Taylor, Carey McGilliard, and Bridget Ferris available at http://wiki.cbr.washington.edu/qerm/sites/qerm/images/1/16/Filled.contour3.R.

2.3 Optional R Packages

1. Alternative methods proposed in Subheading 4 use the "fields" and "rgl" packages hosted by the CRAN mirror sites and "EBImage" bioconductor package. They can be installed using the following instructions:

```
> install.packages("fields")
> install.packages("rgl")
> source("http://bioconductor.org/biocLite.R")
> biocLite("EBImage")
```

2.4 Sample Datasets

The description of methods in Subheadings 3 and 4 was performed on *P. infestans* datasets described in [2]. Pre-processed files, as used in the following Subheading 3, can be downloaded from the Figshare repository using the link http://dx.doi.org/10.6084/m9.figshare.707329. This archive includes:

1. The "Mygtf.gtf" file containing the final transcript calls for *P. infestans* genome version 2.4. The original file is available at http://www.broadinstitute.org/annotation/genome/phytophthora_infestans/MultiDownloads.html. The file "Mygtf.gtf" imported in **step 3** of Subheading 3.1 is a modified version in which lines were sorted in ascending order on the seqname and start position columns according to **Note 3**.
2. The "RXLR_FIRs.csv" file containing intergenic regions length for RXLR effector genes only, imported in **step 10** of Subheading 3.5.
3. The "Tom_expression.csv" file containing gene induction values at 2 days post inoculation on tomato, imported in **Note 7**.

3 Methods

The following methods describe the procedure to draw genome architecture heatmaps and overlay them with scatter plots in R. An alternative method for calculating intergenic region length (Subheading 3.1) in perl is proposed in **Note 1**.

3.1 Calculation of Flanking Intergenic Regions

1. Copy the file "getFeat2.R" into your R working directory. Specify the use of this file as the source for the "getFeat2" function as follows:

```
> source("getFeat2.R")
```

   You may want to ignore genes at the ends of contigs in the analysis. In this case the "getFeat2b" function should be used instead of "getFeat2". *See* **Note 2** for details on how to use it.

2. Load libraries required for this section.

```
> library(rtracklayer)
> library(GenomicRanges)
> library(Rsamtools)
```

3. Place a copy of the .gff file containing feature coordinates for the genome to analyze in your R working directory. Files in the .gff or .gtf format are accepted as input. They must contain either "gene," "exon" or "start codon," and "stop codon" features for the "getFeat2" function to work (*see* **Note 3**). Import the gff file using the "import.gff" function of the GenomicRanges package, replacing "Mygtf.gtf" by the name and extension of the file to analyze.

```
> gff<- import.gff("Mygtf.gtf", asRangedData
=FALSE)
```

4. Generate a Range object called "gffgene" containing gene features using the "getFeat2" function. Specify the type of the file imported ("gff" or "gtf") after the "format=" parameter.

```
> gffgene<-getFeat2(x=gff, format="gtf",
range_types=c("gene"))
```

5. Restore the gene strand information lost after processing using the "getFeat2" function.

```
> strand(gffgene)<-mcols(gffgene)$score
```

6. Clean up the temporary strand information column.

```
> mcols(gffgene)$score<-NULL
```

7. Generate a Range object called "gffintg" containing intergenic region features using the "getFeat2" function. Specify the type of the file imported ("gff" or "gtf") after the "format=" parameter. This operation can take some time to complete.

```
> gffintg<-getFeat2(x=gff, format="gtf",
  range_types=c("intergenic"))
```

8. Create a data frame object called "length_intg" containing one column with intergenic regions indexes and another column with intergenic regions length.

```
> length_intg<-
  as.data.frame(cbind(seq(1:length(ranges(gffi
  ntg))), as.numeric(mcols(gffintg)$length)))
```

9. Rename columns of the "length_intg" data frame as "index" and "length" for convenience in future references.

```
> colnames(length_intg)<-c("index", "length")
```

10. Create a list called "three_intg_index" of indexes for intergenic regions following every gene (located at the 3′ end of each gene), using the "precede" function of the GenomicRanges package.

```
> three_intg_index<-precede(gffgene, gffintg)
```

11. Create a list called "five_intg_index" of indexes for intergenic regions preceding every gene (located at the 5′ end of each gene), using the "follow" function of the GenomicRanges package.

```
> five_intg_index<-follow(gffgene, gffintg)
```

12. Create a data frame object called "gene_data" containing columns with gene identifiers, gene strand, five prime, and three prime intergenic region indexes.

```
> gene_data<-
  as.data.frame(cbind(as.character(mcols(gffg
  ene)$group), as.character(strand(gffgene)),
  as.numeric(five_intg_index),      as.numeric
  (three_intg_index)))
```

13. Rename columns of the "gene_data" data frame as "geneid," "strand," "FivePrime_index," and "ThreePrime_index" for convenience in future references.

```
> colnames(gene_data)<-c("geneid", "strand",
"FivePrime_index", "ThreePrime_index")
```

14. Merge "gene_data" and "length_intg" data frames into a "tempdata" data frame using indexes for five prime intergenic regions as a common data column between the two data frames. The "all.x=TRUE" parameter allows lines of the "gene_data" data frame to be kept with no five prime intergenic region index.

```
> tempdata<-merge(x=gene_data, y=length_intg,
       by.x="FivePrime_index", by.y="index",
   all.x=TRUE)
```

15. Rename columns of the "temp_data" data frame as "delete1", "geneid", "strand", "ThreePrime_index," and "fiveprime" to avoid conflicts with the following steps of the process.

```
> colnames(tempdata)<-c("delete1", "geneid",
"strand", "ThreePrime_index", "fiveprime")
```

16. Merge "tempdata" and "length_intg" data frames into a "FIRdata" data frame using indexes for three prime intergenic regions as a common data column between the two data frames. The "all.x=TRUE" parameter allows lines of the "gene_data" data frame to be kept with no three prime intergenic region index.

```
> FIRdata<-merge(x=tempdata, y=length_intg,
    by.x="ThreePrime_index",   by.y="index",
 all.x=TRUE)
```

17. Clean up unnecessary columns in the "FIRdata" data frame object.

```
> FIRdata$ThreePrime_index<-NULL
> FIRdata$delete1<-NULL
```

18. Rename columns of the "FIRdata" data frame as "geneid," "strand," "fiveprime," and "threeprime" for convenience in future references. The FIRdata table can be exported for external use (*see* **Note 4**).

```
> colnames(FIRdata)<-
      c("geneid", "strand", "fiveprime", "threeprime")
```

3.2 Bin Breaks Setup

1. Set the number of bins to use by replacing "40" in the code line below. This number is stored under the variable name "NumBins." We recommend values between 10 and 60 for an appropriate representation of most eukaryotic genomes.

```
> NumBins=40
```

2. The size and breaks of bins will be calculated based either on five prime or three prime intergenic regions length, depending on which list contains the longest intergenic region of the genome. The corresponding list is stored under the variable name "FIR2Bin"

```
> if ((max(FIRdata$fiveprime, na.rm=TRUE)>max
(FIRdata$threeprime, na.rm=TRUE)) == TRUE)
{
FIR2Bin<-FIRdata$fiveprime
} else {
FIR2Bin<-FIRdata$threeprime
}
```

3. Remove intergenic regions of length 0 or with no length assigned from the "FIR2Bin" list.

```
> FIR2Bin=FIR2Bin[which(FIR2Bin!=0)]
> FIR2Bin<-na.omit(FIR2Bin)
```

4. Determine the number of entries "BinSteps" per bin to separate all entries in the "FIR2Bin" list into a number of quantiles equal to "Numbins-1"

```
> BinSteps<-round(length(FIR2Bin)/(Num Bins-1),
  digits=0)
```

5. Sort the length of intergenic regions stored in the "FIR2Bin" list in ascending order; store the ordered list under the variable name "FIR2BinOrd".

```
> FIR2BinOrd<-sort(FIR2Bin)
```

6. Determine the temporary bin breaks stored based on intergenic region lengths, the position of which delimits quantiles of length "BinSteps." The list of temporary bin breaks is stored under the list named "TempBinLimits."

```
> TempBinLimits<-FIR2BinOrd[seq(FIR2BinOrd[2*
  BinSteps],
       length(FIR2BinOrd),BinSteps)]
```

7. To allow a more complete coverage of the range of intergenic region length present in the genome analyze, the maximum value in the list "FIR2Bin" is added as the last break in the list "TempBinLimit".

```
> TempBinLimits[length(TempBinLimits)+1]<-
      max(FIR2Bin, na.rm=TRUE)
```

8. Fit an exponential distribution to the values of temporary bin breaks in "TemBinLimits" list using the non-linear regression function "nls." A prevalent weight is given to the extreme values in the "TempBinLimits" list during the regression to ensure a better representation of genes with extreme intergenic region length (*see* **Note 5**).

```
> x<-seq(length(TempBinLimits))
> fit<-nls(log(TempBinLimits) ~ a*x + b, start
= c(a=0, b=0),
     algorithm='port',weights=((x-0.5
  * NumBins)^2))
```

9. Calculates the predicted bin breaks based on the exponential regression and store values in the "BinLimits" list.

```
>pred=predict(fit, x)
>BinLimits=c(1, round(exp(pred),0), max(FIR2Bin))
```

Comparisons of the architecture of several genomes, using the same set of bin breaks might be required. **Note 6** describes how to export the list of bin breaks calculated here and how to import an external list of bin breaks.

3.3 Data Binning

1. Assign five prime and three prime intergenic regions of every gene to bins delimited by breaks in the "BinLimits" list. Assigned bins are stored under variables called "xbin" and "ybin" for five prime and three prime intergenic regions, respectively.

```
>xbin=cut(FIRdata$fiveprime,breaks=c(BinLimits))
>ybin=cut(FIRdata$threeprime,breaks=c(BinLimits))
```

2. Concatenate the list of assigned bins to the "FIRdata" data frame. This step also concatenates a list of gene-associated values called "genevalue" that is set to "1" for every gene. A custom list of gene-associated values can be provided instead (*see* **Note 7**).

```
> FIRdata<-cbind(FIRdata, xbin, ybin,
      genevalue=rep(1,length(FIRdata$fiveprime)))
```

3. Create the matrix "GenValMatrix" containing the number of genes in each bin. The number of genes is calculated by summing up gene-associated values of 1 for each gene, using the function "sum." Other operations can be performed on gene-associated values using different built-in or custom functions (*see* **Note 8**). **Note 9** describes how to export the matrix calculated here or how to import an external matrix of values.

```
>GenValMatrix<-with(FIRdata, tapply(genevalue,
list(xbin, ybin), sum))
```

3.4 Heatmap Drawing

1. Set units of the graph axes in the three dimensions using the number of columns in "GenValMatrix", the number of rows in "GenValMatrix," and the maximum value in "GenValMatrix" for *x*, *y*, and *z* dimensions, respectively.

```
>x<-1:ncol(GenValMatrix)
>y<-1:nrow(GenValMatrix)
>zlim = range(as.numeric (unlist(GenVal
Matrix)) , finite=TRUE)
```

2. Set the color palette "mypalette" using the "colorRampPalette" function. The number and identifiers of colors can be freely chosen to modify the aspect of the heatmap, by changing the list proposed below.

```
> mypalette<-colorRampPalette(c( "white",
"darkblue", "forestgreen", "goldenrod1",
"orangered", "red3", "darkred"), space="rgb")
```

3. Create a list of colors "mycol" based on the "mypalette" color palette and the number of subdivisions "mynlevels."

```
>mycol=mypalette(2*max(GenValMatrix, na.rm=TRUE))
```

4. Create a list of labels "mylabels" for axes by concatenating the length of the shortest and longest intergenic regions in each bin (stored in the "BinLimits" list) separated by a hyphen.

```
> mylabels<-paste(BinLimits[1:length(BinLim
its)-1], BinLimits[2:length(BinLimits)], sep="
- ", collapse=NULL)
```

5. Perform heatmap graph rendering using the "filled.contour" function (Fig. 2). 3D rendering of the "GenValMatrix" matrix is also possible (*see* **Note 10**).

```
> filled.contour(x, y, z=GenValMatrix,
   plot.title = title(main ="Phytophthora infestans
genome",
   xlab = "five prime intergenic regions",
   ylab = "three prime intergenic regions",
   cex.main=0.8, cex.lab=0.5),
   key.title = title(main ="Number of
   genes", cex.main=0.5,
                line=1),
   col=mycol,
   levels = pretty(zlim, 2*max(GenValMatrix,
na.rm=TRUE)),
   plot.axes={axis(1,at=x, labels=mylabels, las=2,
               cex.axis=0.5);
            axis(2,at=y,  labels=mylabels,
         cex.axis=0.5)})
```

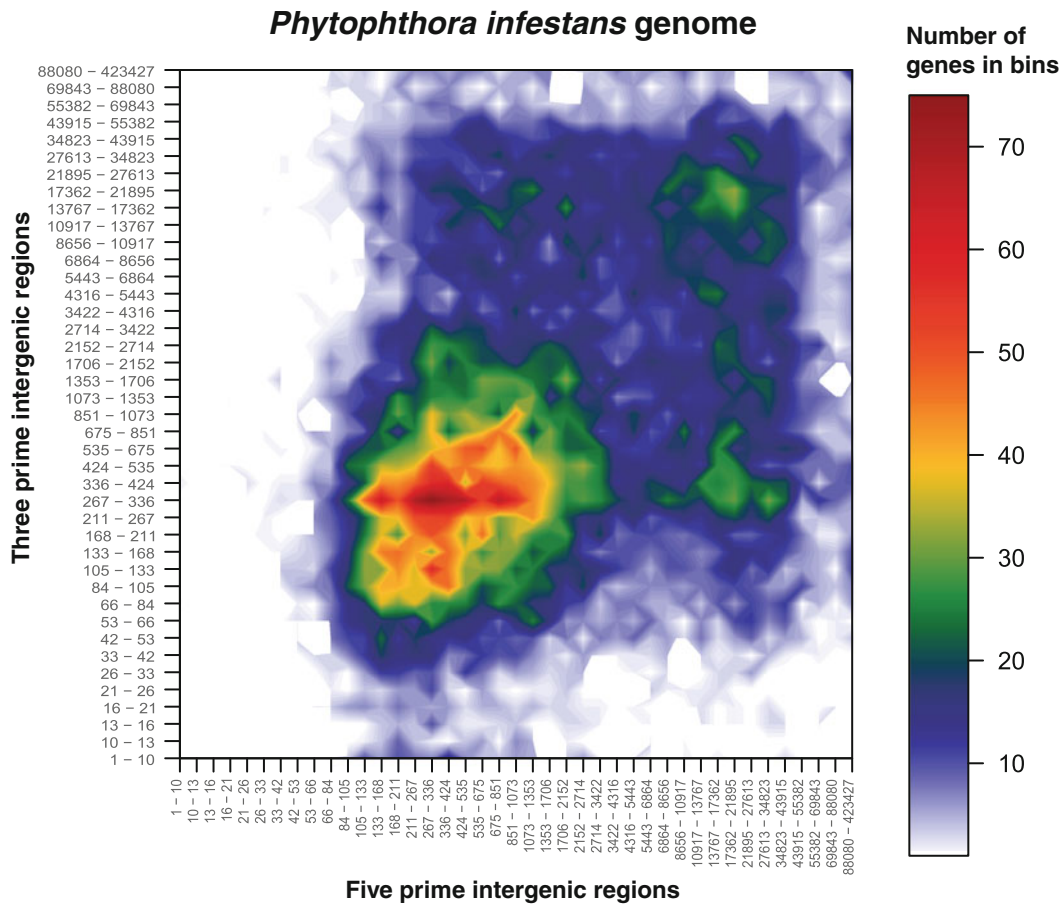

Fig. 2 The heatmap plot output of **step 5** of Subheading 3.4, representing gene counts in *P. infestans* genome. The initial gtf file was obtained at http://www.broadinstitute.org/annotation/genome/phytophthora_infestans/MultiDownloads.html. Minor edits were performed in Adobe Illustrator to include italic fonts in the main title and improve the legibility of the color key and axis labels

3.5 Overlaying a Scatter Plot Over a Genome Architecture Heatmap

1. Copy the code of the "filled.contour3" function given in Subheading 2 into a text file named "filled.contour3.R" and place this file in your R working directory. Specify the use of this file as the source for the "filled.contour3" function as follows:

```
> source('filled.contour3.R')
```

2. Load libraries required for this section.

```
> library(png)
> library(gridExtra)
> library(ggplot2)
```

3. *See* **Note 11** if you want to re-use a scatter plot background image generated from a previous analysis. Automatically set the name of the heatmap image file to be used as a background for the scatter plot. The name is composed of the system date (year, month, and day) and time (hours, minutes, seconds)

followed by "_graph" and stored in the variable "image_name" for future reference.

```
> image_name<-paste(as.character(format(Sys.time(),
        "%Y%m%d%H%M%S")), "_graph", sep="")
```

4. Create a .png image file in your R working directory named after the "image_name" variable. For alternative image file formats *see* **Note 12**.

```
> png(filename = paste(image_name, ".png", sep=""))
```

5. Remove margins of the graph so that it fully covers the background of the scatter plot to be overlaid.

```
> par(mar=c(0,0,0,0))
```

6. Perform heatmap graph rendering using the "filled.contour3" function. This modified version of the "filled.contour" function does not draw the color key scale allowing the heatmap to fully cover the graphic device area.

```
> filled.contour3(x, y, z=GenValMatrix,
     col=mycol,
     levels = pretty(zlim, 2*max(GenValMatrix,
  na.rm=TRUE)),
     frame.plot = FALSE,
     axes = FALSE)
```

7. Save current graph to the file specified in step 4 and close the graphic device.

```
> dev.off()
> quartz.save("heatmap.pdf", type="pdf")
```

8. Load the heatmap background image created previously in a variable called "img" using the "readPNG" function of the "png" package.

```
> img <- readPNG(paste(image_name, ".png", sep=""))
```

9. Convert the background image into a grid graphical object using the "rasterGrob" function of the "gridExtra" package. This object is stored under the name "g."

```
> g <- rasterGrob(img, interpolate=TRUE)
```

10. Import data to be shown as a scatter plot. In the example below, a comma delimited table containing five prime and three prime intergenic regions for RXLR effector genes (called "RXLR_FIRs.csv") is imported using the "as.data.frame" and "read.csv" functions. Note that the file should be located in the R working directory to allow the omission of the filepath when calling the file.

```
> rxlrData<-as.data.frame(read.csv('RXLR_FIRs.csv',
header=TRUE))
```

11. Plot the "rxlrData" data as a scatter plot over the grid object "g" corresponding to the background heatmap image using the "ggplot" function of the "ggplot2" package. After "aes(",

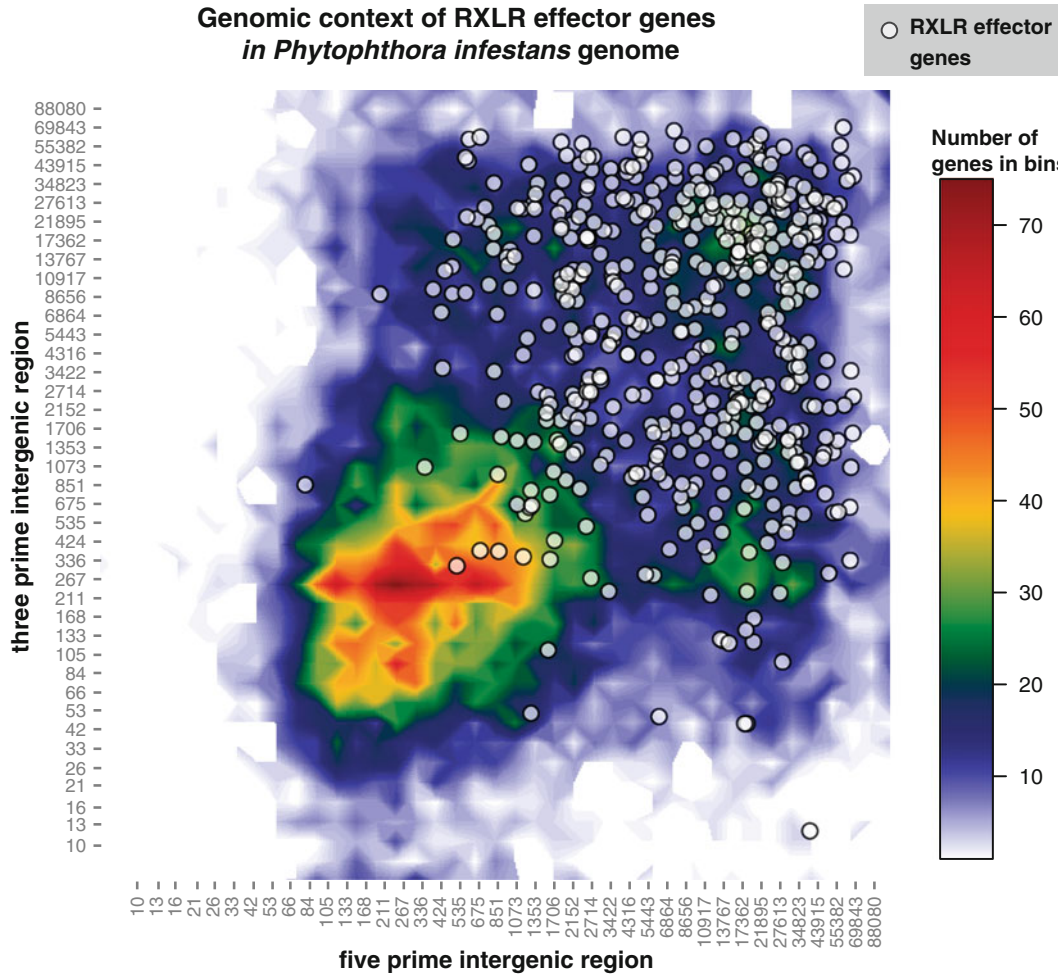

Fig. 3 Overlay of a scatter plot showing the genomic context for RXLR effector genes on the heatmap of *P. infestans* genome architecture. The figure shows the output of **step 10** of Subheading 3.5 with edits performed in Adobe Illustrator to display italic fonts in the main title and include a color key and scatter plot legend

the "x=" and "y=" arguments specify data to use as 5′ and 3′ FIRs, respectively. This should be provided as the name of the variable referring to the dataset, followed by "$" and the name of the column containing either 5′ or 3′ FIR length data ("rxlrData$rxlr_five" and "rxlrData$rxlr_three, respectively in the example below). The following options are used to format the graph: "annotation_custom" specifies the object "g" as a background image and sets its dimension to the whole plot area; ""coord_fixed" locks the x and y ratio to 1 so that the background always covers the whole plot area; "geom._point" specifies the style of dot to use for the scatter plot; "scale_y_ log10" and "scale_x_log10" set axes in logarithm scale and

define their limits; "theme(axis" formats axes labels and titles. The resulting graph is shown in Fig. 3. A warning message may inform you of data points outside the scatter plot range that are not shown.

```
> ggplot(data=rxlrData,
    aes(x=rxlrData$rxlr_five, y=rxlrData$rxlr_three,
          geom="blank")) +
    annotation_custom(g, xmin=-Inf, xmax=Inf, ymin=
 -Inf,
          ymax=Inf) +
    coord_fixed(ratio=1) +
    geom_point(shape=21,fill="white",colour="black",
          size=4, alpha=0.7, na.rm=FALSE) +
    scale_y_log10(breaks = BinLimits[2:length(BinLimits)],
          limits = c(BinLimits[2], BinLimits
       [NumBins +1])) +
    scale_x_log10(breaks=          BinLimits
 [2:length(BinLimits)],
          limits=c(BinLimits[2],
       BinLimits[NumBins +1])) +
    theme(axis.text.y=element_text(size = 10,
 vjust=0.5))    +    theme(axis.text.x=element_
 text(size=10, vjust=0.5,
          angle=90)) +
    theme(axis.title.x = element_text(face=
    "bold",size=12)) +
    xlab("five prime intergenic region") +
    theme(axis.title.y      =      element_
    text(face="bold",size=12)) + ylab("three
    prime intergenic region")
```

4 Notes

1. For the calculation of intergenic region length in perl (Subheading 3.1), the "Calculate_FIR_length.pl" script available from the Figshare repository using the link http://dx.doi.org/10.6084/m9.figshare.707328. This script takes either .gff3 or .gtf files as input. It sets the values of FIRs at the border of contigs to "NA" similar to the "getFeat2b" R function described in **Note 2**. To run it, use:

   ```
   > perl Calculate_FIR_length.pl
   ```

   Several prompts will invite the user to select appropriate options. First, type in "gff3" or "gtf" to specify whether the input file is in .gff3 or .gtf format:

   ```
   Is the file in gff3 or gtf format? (gff3/gtf)
   ```

   Second, type in the name and extension of the input file (e.g., "Myfile.gff3")

```
Please enter the name of the gff or gtf file:
```

Third, type in "gene", "mRNA," or "exon" to specify the type of feature included in the input file to use as gene models:

```
Which feature/type to process? (gene/mRNA/exon):
```

Finally, enter the name of the output file, including the ".csv" extension (e.g., "MyFIRs.csv")

```
Enter name of output file:
```

This will produce a comma delimited file containing FIR lengths (stored in the folder where the script was executed). To import this file in R and continue with steps under Subheading 3.2 of the analysis, use:

```
> FIRdata<-read.csv(file="MyFIRs.csv", sep=",")
```

2. By default, the "getFeat2" function calculates distance to the end of contigs as the FIRs for genes residing at the edge of contigs. Because contig edges are often difficult to assemble, these FIRs are likely underestimated. As an alternative, you can use the "getFeat2b" function to set FIRs at the edge of contigs to "NA," and therefore ignore them in subsequent steps of the analysis. To do so, copy the file "getFeat2b.R" into your R working directory. Specify to use this file as the source for the "getFeat2b" function as follows:

```
> source("getFeat2b.R")
```

Then at **steps.4** and 7 of Subheading 3.1 call the "getFeat2b" function instead of "getFeat2" as follows:

```
> gffgene<-getFeat2b(x=gff,format="gff", range_
types=c("gene"))
> gffintg<-getFeat2b(x=gff, format="gff",
        range_types=c("intergenic")
        )
```

3. The methods described here take .gff and .gtf files as input. Gene and intergenic regions' features are generated using the custom "getFeat2" function, a modified version of the "getFeat" function written by T. Girke available at http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/gffMod.R. The file type "gff" should be selected when the annotation file contains gene features, and "gtf" file type should be selected otherwise. If the wrong file type is selected, the following error message might show:

```
Error in .Call2("solve_user_SEW0", start, end,
width, PACKAGE = "IRanges") : solving row 1: range
cannot be determined from the supplied arguments
(too many NAs)
```

To avoid errors while running "getFeat2", the annotation file should not contain any blank lines and must be sorted by

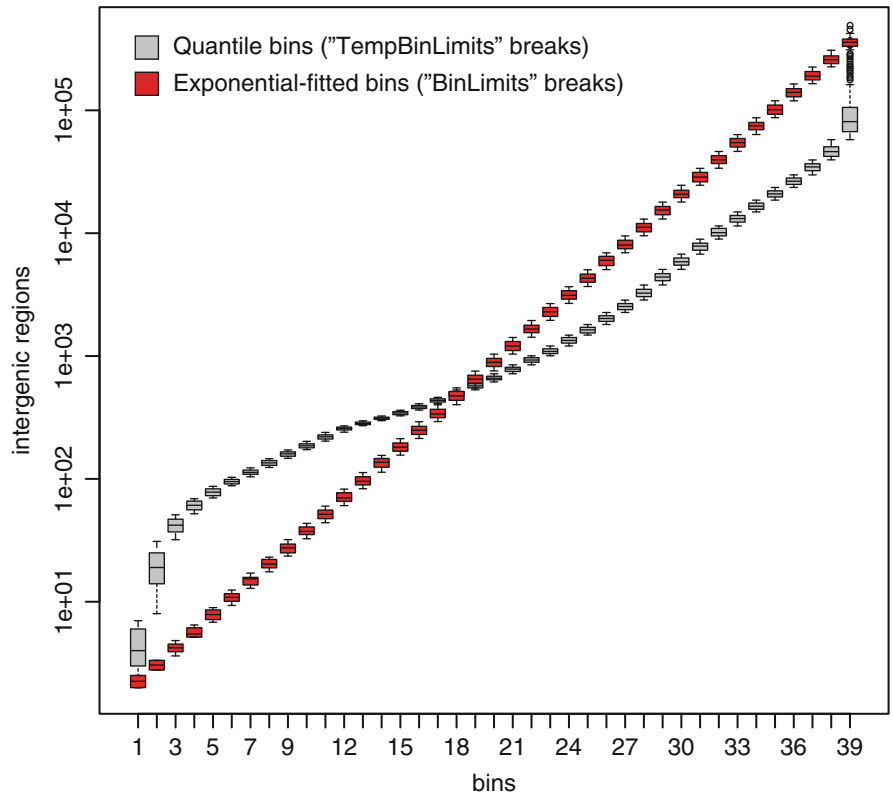

Fig. 4 Distribution of *P. infestans* intergenic regions length in temporary quantile bins and exponential-fitted bins

ascending order on the seqname (first column) first and then start position (third column).

4. To export the “FIRdata” table as a comma delimited table (.csv file, exported to R working directory by default), the following instruction can be used:

```
> write.table(FIRdata,file="MyFIRs.csv", sep=",",
row.names=FALSE)
```

To import a .csv file to start the analysis directly at Subheading 3.2 use:

```
> FIRdata<-read.csv(file="MyFIRs.csv",sep=",")
```

5. A comparison of temporary bins based on quantiles (with breaks defined by “TempBinLimits”) and bins derived from exponential regression (with breaks defined by “BinLimits”) allows to control whether the distribution of intergenic regions length in bins has been appropriately optimized. The following code creates a boxplot representation of data in each bin, for temporary (grey) and final bins (red), as shown in Fig. 4. It

required installation of the "fields" package to run. On *P. infestans* intergenic region data, a significant reduction in the variance in bins is obtained with the proposed exponential regression. Alternative regression models may be considered in step 8 of Subheading 3.2 for some datasets.

```
> x<-seq(length(TempBinLimits))
> library(fields)
> bplot(FIR2Bin, as.numeric(cut(FIR2Bin,
    breaks=TempBinLimits)),
    breaks=TempBinLimits, xlim=c(1,length(T
    empBinLimits)), col="grey", log="y",
    xlab="bins", ylab="intergenic regions")
> par(new=TRUE)
> bplot(FIR2Bin, as.numeric(cut(FIR2Bin, breaks
=BinLimits)),
    breaks=BinLimits, xlim=c(1,length(BinLimits)),
    col="red", log="y", axes=FALSE, ann=FALSE)
```

6. To export the "BinLimits" list as list in a text file (.txt file, exported to R working directory by default), the following instruction can be used:

```
> write.table(BinLimits,file="MyBins.txt")
```

To import an external set of bin breaks saved as a .txt file, use:

```
> BinLimits<-
as.numeric(unlist(read.table(file="MyBins.txt",
            header=TRUE, row.names=1)))
```

To start the analysis directly from Subheading 3.3, both "FIRdata" and "BinLimits" must be imported as described in **Notes 4** and **6,** respectively.

7. Typical gene-associated values are gene expression values or number of SNPs per genes. The following example illustrates how to replace the gene-associated value column in the "FIRdata" table by imported gene expression data using the "merge" function. Datasets used in this example are available in the Figshare repository using the link http://dx.doi.org/10.6084/m9.figshare.707329.

```
> GeneExpression<-
as.data.frame(read.csv('Tom_expression.csv',
        header=TRUE)) #import gene expression data
> colnames(GeneExpression)<-c("geneid", "genevalue")
        #rename columns in gene expression
    data frame
> FIRdata$genevalue<-NULL
        #delete former gene associated value
> tempdata<-as.data.frame(FIRdata)
    # convert to data frame to perform merge
    function
> FIRdata<-merge(x=tempdata, y=GeneExpression,
        by.x="geneid", by.y="geneid", all.
    x=TRUE)
```

8. Virtually any function can be applied on gene-associated values, the function "mean" being among the more useful when analyzing gene expression data. The following creates a "GenValMatrix" containing average gene expression in bins using data imported in **Note 7**.

```
> GenValMatrix<-with(FIRdata, tapply(genevalue,
  list(xbin, ybin), mean))
```

Functions such as "mean" introduce significant noise in the heatmap due to bins containing few genes. It is therefore advisable to ignore bins with too few elements. The following lines create a "CountMatrix" containing the number of genes in bins that is used to replace the gene expression values of bins with less than four genes by "0" in the GenValMatrix.

```
> CountMatrix<-with(FIRdata, tapply(genevalue,
                list(xbin, ybin), length))
> GenValMatrix[CountMatrix <= 4]<-0
```

The color scale may need to be modified to fit with the gene-associated values. In particular, gene expression data contain negative and positive values and the color scale needs to be adjusted so that "0" values are shown with a neutral color (here "white"). The following lines generate the gene expression heatmap shown in Fig. 5.

```
> mypalette<-colorRampPalette(c("black", "black",
"darkblue",
   "blue",    "dodgerblue3",    "dodgerblue",
   "white",     "goldenrod1",     "orangered",
   "red3", "red4", "red4"), space="rgb")
>mycol=mypalette(7+50*(max(GenValMatrix,na.rm=TRUE)-
   min(GenValMatrix, na.rm=TRUE)))
> filled.contour(x, y, z=GenValMatrix,
   plot.title = title(main ="Phytophthora infestans
gene
      induction at 2 days post inoculation on
   tomato",
   xlab = "five prime intergenic regions",
   ylab = "three prime intergenic regions", cex.main=0.8,
      cex.lab=0.5),
   key.title = title(main ="Average gene
   induction",
      cex.main=0.5, line=1),
   col=mycol,
   levels=pretty(zlim,50*(max(GenValMatrix,
na.rm=TRUE)-
          min(GenValMatrix, na.rm=TRUE))),
   plot.axes={axis(1,at=x, labels=mylabels, las=2,
          cex.axis=0.5),
          axis(2,at=y, labels=mylabels, cex.
       axis=0.5)})
```

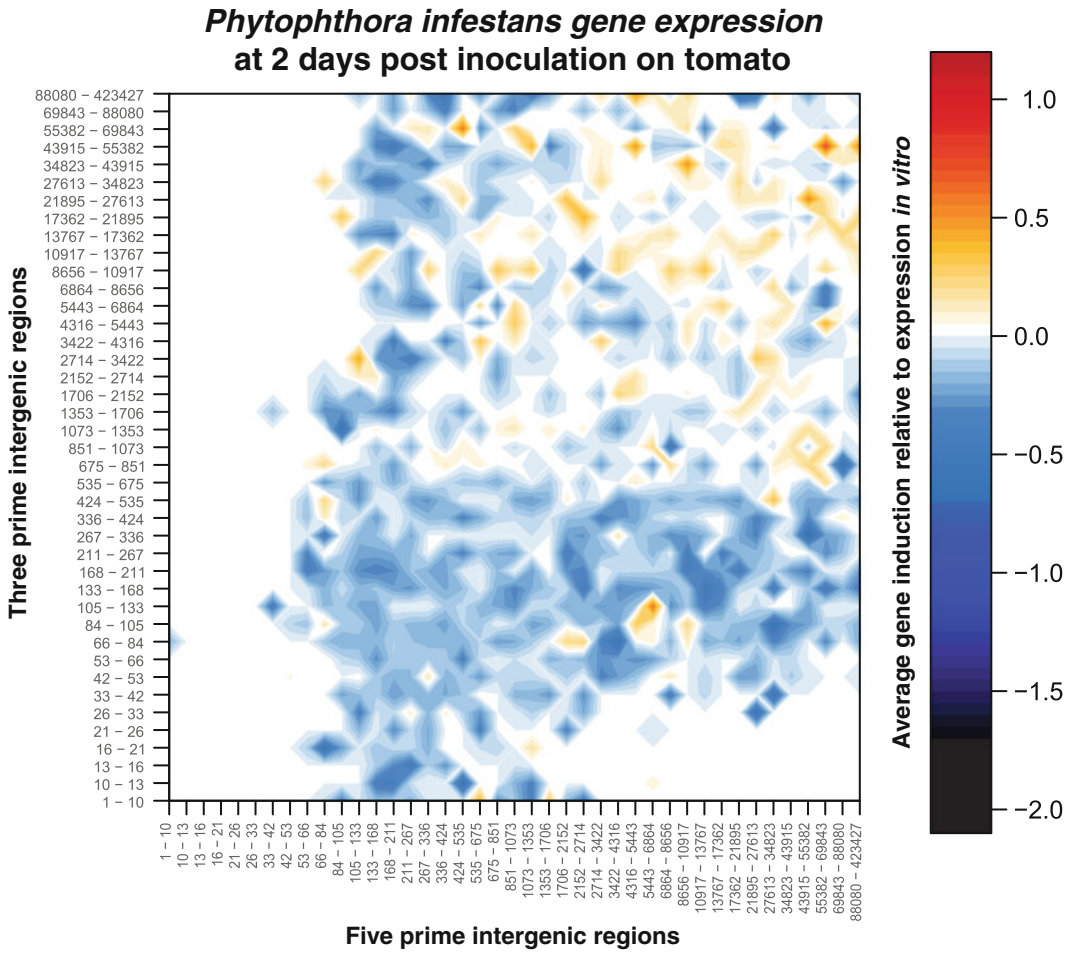

Fig. 5 Heatmap representing the average induction fold at 2 days postinoculation on tomato for *P. infestans* genes, relative to gene expression in vitro. Gene induction fold was averaged per bin and plotted as described in **Note 8**. Minor edits were performed in Adobe Illustrator to include italic fonts in the main title and improve the legibility of the color key and axis labels

9. To export the "GenValMatrix" table as a comma delimited table (.csv file, exported to R working directory by default), the following instruction can be used:

```
> write.table(GenValMatrix, file="MyMatrix.csv", sep=",", row.names=FALSE)
```

To import an external data matrix saved as a .csv file, use:

```
>GenValMatrix<-as.matrix(read.csv(file="MyMatrix.csv", sep=","))
```

To start the analysis directly from Subheading 3.4, both "BinLimits" and "GenValMatrix" must be imported as described in **Notes 6** and **9** respectively.

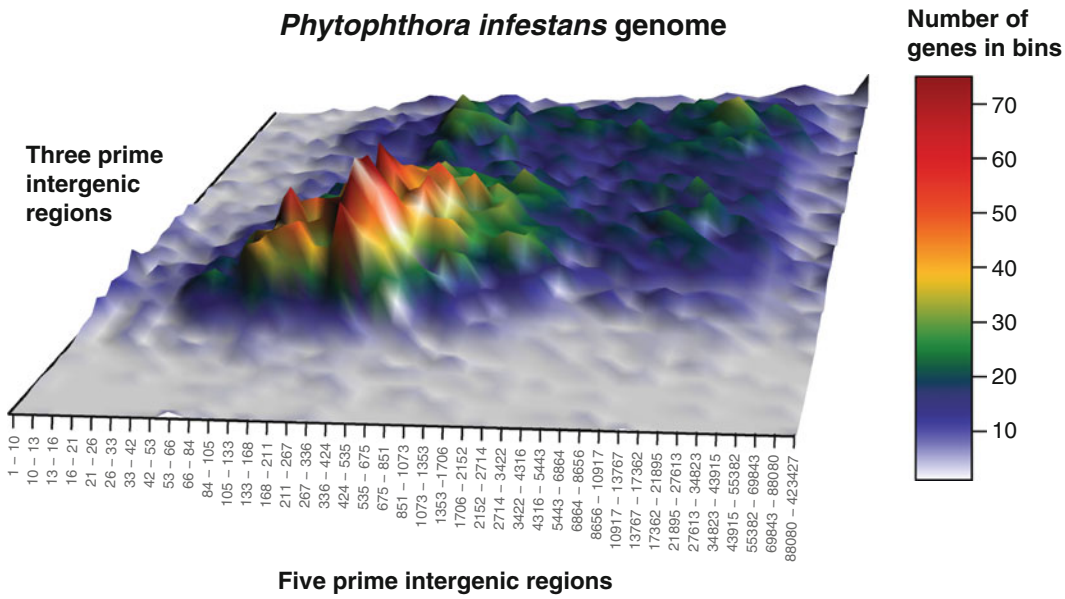

Fig. 6 A 3D view of gene counts in *P. infestans* genome created using RGL package. Legends and axis labels were added in Adobe Illustrator

10. Data in the "GenValMatrix" matrix can be visualized in 3D using the "rgl" package. The following describes how, from **step 6** of the Subheading 3.4 "Heatmap drawing", a 3D real-time visualization of the genome architecture can be obtained (Fig. 6). The "rgl" package must be installed for this method.

```
> library(rgl)                    #load the rgl library
> clear3d("all")                  #clear the graphic device
> rgl.light(theta = 45, phi = 45, viewpoint.
rel=TRUE)
#light parameters
> z<-2*GenValMatrix               #Exaggerate the relief
> z[is.na(z)]<-0     #Replace <NA> by '0'
> x<-20*(1:nrow(z))
> y<-20*(1:ncol(z))   #set row and column spac-
ing to 20
> zlim<-range(z, na.rm=TRUE) #determine maximum z
values
> zlen<-zlim[2]-zlim[1] +1
> col <- mycol[z-zlim[1]+1] #determine       the
matrix of colors
> rgl.surface(x, y, z, color=col, alpha=1,
back="fill")
```

11. To display several gene subsets from the same genome as a scatter plot, a scatter plot background image generated from a previous analysis can be re-used. For this, skip steps 3–7 of

Subheading 3.5 and replace the variable "image_name" by the name of the image file to import in step 8 of Subheading 3.5.

12. File formats JPEG and TIFF are supported for the scatter plot background image. To write the background image file in the chosen format, replace step 4 of Subheading 3.5 with one of the following:

```
> jpeg(filename = paste(image_name, ".jpeg", sep=""))
> tiff(filename = paste(image_name, ".tiff", sep=""))
```

Loading JPEG or TIFF files in R will require the "EBImage" package. Replace **step 8** of Subheading 3.5 by the following, using either ".jpeg" or ".tiff" as the file extension:

```
> library("EBImage")
> img <- readImage(files = paste(image_name, ".
jpeg", sep=""))
```

Acknowledgments

We are grateful to Adam Taranto for comments on drafts of this chapter.

References

1. Koonin EV, Wolf YI (2010) Constraints and plasticity in genome and molecular-phenome evolution. Nat Rev Genet 11:487–498
2. Haas BJ et al (2009) Genome sequence and analysis of the Irish potato famine pathogen *Phytophthora infestans*. Nature 461:393–398
3. Spanu PD, Abbott JC, Amselem J, Burgis TA, Soanes DM, Stüber K, Loren van Themaat EV, Brown JKM, Butcher SA, Gurr SJ (2010) Genome Expansion and Gene Loss in Powdery Mildew Fungi Reveal Tradeoffs in Extreme Parasitism. Science 330:1543
4. Duplessis S, Cuomo CA, Lin YC, Aerts A, Tisserant E, Veneault-Fourrey C, Joly DL, Hacquard S, Amselem J, Cantarel BL (2011) Obligate biotrophy features unraveled by the genomic analysis of rust fungi. Proc Natl Acad Sci 108:9166
5. Raffaele S, Win J, Cano L, Kamoun S (2010) Analyses of genome architecture and gene expression reveal novel candidate virulence factors in the secretome of *Phytophthora infestans*. BMC Genomics 11:637
6. van Damme M, Cano L, Oliva R, Schornack S, Segretin ME, Kamoun S, Raffaele, S (2012) Evolutionary and Functional Dynamics of Oomycete Effector Genes. In: Martin F, Kamoun S (eds) Effectors in plant-microbe interactions. Wiley, pp 103–122
7. Vleeshouwers VGAA, Raffaele S, Vossen JH, Champouret N, Oliva R, Segretin ME, Rietman H, Cano LM, Lokossou A, Kessel G (2011) Understanding and Exploiting Late Blight Resistance in the Age of Effectors. Annu Rev Phytopathol 49:507–531
8. Raffaele S, Kamoun S (2012) Genome evolution in filamentous plant pathogens: why bigger can be better. Nat Rev Microbiol 10:417–430
9. Pain A et al (2008) The genome of the simian and human malaria parasite *Plasmodium knowlesi*. Nature 455:799–803
10. Lévesque CA et al (2010) Genome sequence of the necrotrophic plant pathogen *Pythium ultimum* reveals original pathogenicity mechanisms and effector repertoire. Genome Biol 11:R73
11. Win J, Morgan W, Bos J, Krasileva KV, Cano LM, Chaparro-Garcia A, Ammar R, Staskawicz BJ, Kamoun S (2007) Adaptive Evolution Has Targeted the C-Terminal Domain of the RXLR Effectors of Plant Pathogenic Oomycetes. Plant Cell 19:2349–2369
12. Kemen E, Gardiner A, Schultz-Larsen T, Kemen AC, Balmuth AL, Robert-Seilaniantz A,

Bailey K, Holub E, Studholme DJ, MacLean D (2011) Gene Gain and Loss during Evolution of Obligate Parasitism in the White Rust Pathogen of *Arabidopsis thaliana*. PLoS Biol 9:e1001094

13. Sharma VK, Kumar N, Prakash T, Taylor TD (2012) Fast and Accurate Taxonomic Assignments of Metagenomic Sequences Using MetaBin. PLoS One 7:e34030
14. Xiong X, Xu W, Eberlin LS, Wiseman JM, Fang X, Jiang Y, Huang Z, Zhang Y, Cooks RG, Ouyang Z (2012) Data Processing for 3D Mass Spectrometry Imaging. J Am Soc Mass Spectrom 23:1147–1156
15. Mendhurwar KA, Devabhaktuni VK, Raut R (2008) Binning algorithm for accurate computer aided device modeling. In: International symposium on circuits and systems. pp 2773–2776
16. Raffaele S, Farrer RA, Cano LM, Studholme DJ, MacLean D, Thines M, Jiang RHY, Zody MC, Kunjeti SG, Donofrio NM (2010) Genome Evolution Following Host Jumps in the Irish Potato Famine Pathogen Lineage. Science 330:1540