Genome Visualization with Circos

Session 3 — Links, Bundles and Track Automation

Martin Krzywinski
Genome Sciences Centre
100-570 West 7th Ave
Vancouver BC V5Z 4S6 Canada
1-604-877-6000 x 673262
martink@bcgsc.ca
http://mkweb.bcgsc.ca

Circos
http://circos.ca

Course Materials
http://circos.ca/documentation/course

Genome Sciences Center
http://bcgsc.ca

Version History
v0.20 3 May 2016
v0.19 12 May 2014
v0.18 6 May 2014
v0.17 11 Jun 2011
v0.16 23 Jul 2010
v0.15 1 Jul 2010
v0.14 30 Jun 2010
v0.13 30 Jun 2010
v0.12 29 Jun 2010
v0.11 29 Jun 2010
v0.10 21 Jun 2010

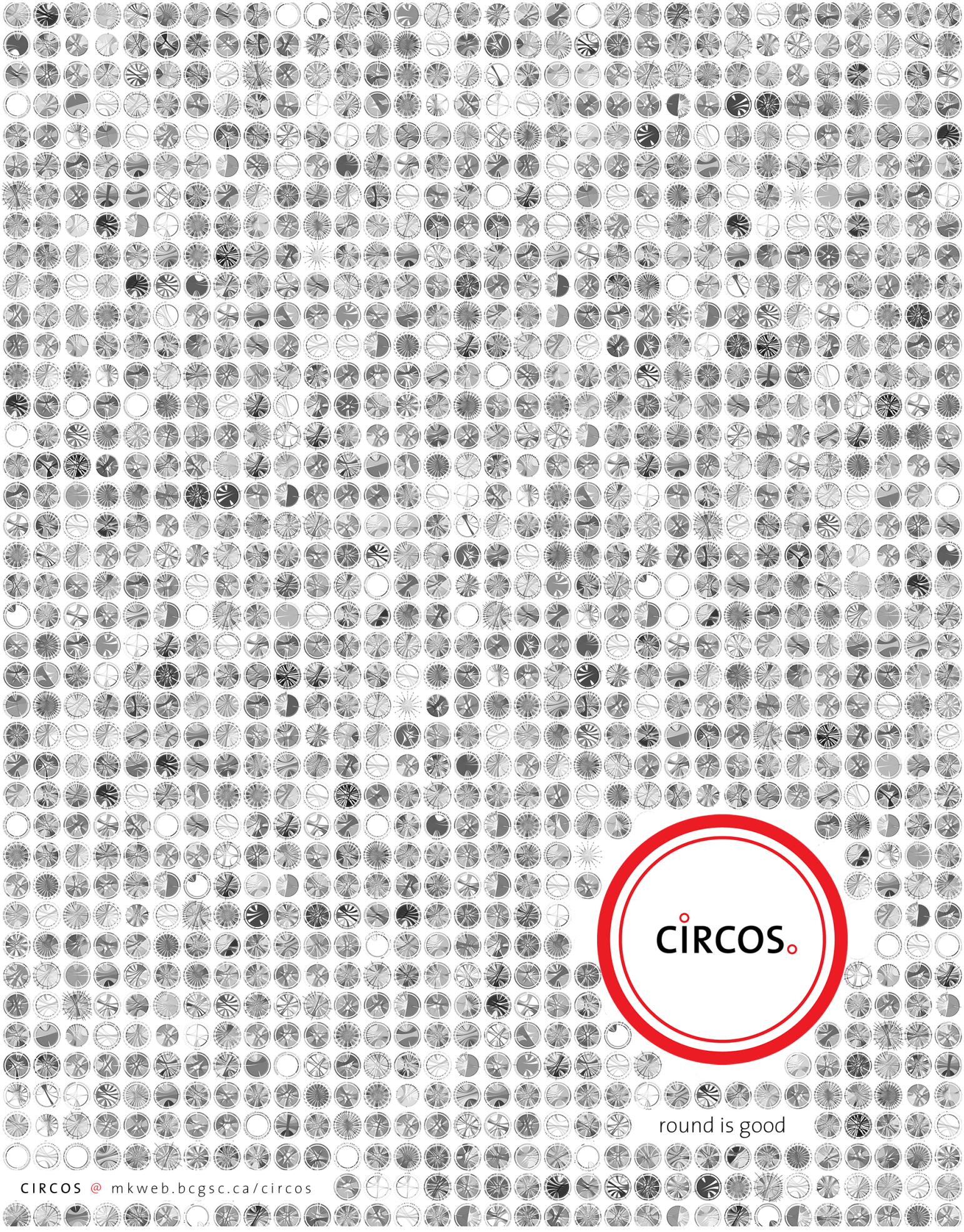
CIRCOS.
round is good
CIRCOS @ mkweb.bcgsc.ca/circos

Table of Contents

Introduction to Links and Rules

Links are the raison d'être of Circos. This track type allows you to connect two positions of any ideograms with Bezier curves. A variety of parameters allow you to change the curve geometry, turn curves into ribbons and control the ribbon twists.

Rules are blocks that modify the format of data points (e.g. scatter plot points, histogram bins, etc) or links. Using rules you can adjust how a data point (or link) is display by altering its color, shape, geometry, transparency or visibility.

Figure 1 shows examples of how links and rules can be combined. Figure 1A shows a large number of links between three ideograms. In Figure 1B, the links are given a default grey color and rules have been used to turn links that start near specific regions red and black. In Figure 1C, rules were used to change the geometry and color of links that connect nearby intra-chromosomal region to make these links blue and point outwards. Rules in Figure 1D were used to adjust the thickness and color of the links.

Rules are extremely useful in creating texture in the links, which can be very dense and difficult to otherwise interpret. By changing link color, geometry and transparency, based on link position and/or size, rules allow you to quickly draw focus to specific parts of the data set without having to adjust the input data files.

Links that connect large regions are best drawn as ribbons, as shown in Figure 2. Thus, instead of using rules to adjust the thickness of the link line, as shown in Figure 1D, turning the links into ribbons automates this process. Furthermore, whereas line links always have constant thickness, the thickness of a ribbon link varies across the length of the link.

In this lesson we will create a figure that compares human chromosome 1 to the entire mouse genome, showing synteny between these regions.

All the data tracks for this session can be created using `3/data/create.tracks`. The same raw UCSC Genome Browser download data is used for this session as for session 2.

Lesson 1 – Ideogram, Tick, Grid and Label Layout

In this lesson, we'll setup the ideogram, tick and grid layout for the rest of the images in this session. Using the combined human and mouse karyotype file, we select only the mouse chromosomes `mm1-mm19` and human chromosome 1 (`hs1`) to be displayed. This is done using the `chromosomes` parameter.

```
karyotype = ../../data/karyotype/karyotype.human.hg19.txt,
            ../../data/karyotype/karyotype.mouse.mm9.txt

chromosomes_units           = 1000000
chromosomes_display_default = no

chromosomes          = hs1[a]:0-120;hs1[b]:140-);/mm\d/
chromosomes_order    = mm19,b,a
chromosomes_reverse  = a,b
chromosomes_scale    = a:0.25r,b:0.25r

<highlights>
<highlight>
show = yes
file = ../data/highlight.txt
r0   = 0.99r
r1   = 0.999r
</highlight>
</highlights>

<<include ../etc/ideogram.conf>>
<<include ../etc/ticks.conf>>
<<include ../etc/image.conf>>

<<include etc/housekeeping.conf>>

# ../etc/image.conf
<image>
angle_orientation* = counterclockwise
</image>

<<include ../../etc/image.conf>>
```

We'll reverse the scale of all the mouse chromosomes using `chromosomes_reverse` so that both genomes start at the top of the figure.

The highlights block provides the color index for the mouse chromosomes. We're using the UCSC Genome Browser color convention, but applying it to mouse chromosomes. The result is shown in Figure 3.

To depict the details of information on human chromosome 1 at higher resolution, we use a relative scale `0.5r` to make it occupy 50% of the figure.

As before, the thickness of the ideograms is kept small (15 pixels, which is 1.5% of the figure width). The ideograms play to role of both axes and scale, and are used for navigation. The ideograms should be visible, but also be subtle, as not to draw attention away from the data.

The tick marks have been designed to accommodate the increased scale of hs1. Mouse chromosomes have 50Mb ticks and the human chromosome has additional 5Mb and 1Mb ticks.

Lesson 2 – Drawing Links

In this lesson, you will see how to create links. Links are represented by a pair of genomic regions. The link is used to connect these regions with a line, or a ribbon. Each of the regions can be of any size, but a region is confined to a single chromosome.

Links are defined in `<link>` blocks which belong to an outer `<links>` block. The `radius` parameter specifies the radial position of the ends of the link and the `bezier_radius` parameter gives the radial position of the control point for the Bezier curve. The smaller the `bezier_radius`, the closer the link comes to the center of the circle. Note that the Bezier curve does not generally pass through its control point—the point is used to define the tangent of the curve at its ends.

```
<links>
<link>
file          = ../data/links.txt
bezier_radius = 0r
radius        = 0.85r
thickness     = 1p
color         = black_a5
</link>
</links>
```

Drawing the links as a 1 pixel thick line and using black with transparency (`black_a5`), the result is shown in Figure 4. Using transparent lines for links is a very good idea when the link data set is dense. Compare Figure 4, where transparency is used, with Figure 5, where each link is an opaque black line. The latter is very difficult to interpret.

WHAT IS SHOWN IN THE FIGURE?

For each 2 Mb window on hs1, the figure shows the top 20 alignments between hs1 and mouse chromosomes 1-19. The link data is created using the `makelinks` script which is found in `sessions/3/data/ucsc`.

```
> cd session/3/data
> cat ../../2/data/ucsc/hg18.mm.chr1.txt | egrep -v -i "(random|x|y|chrm)" |
      ../../2/data/ucsc/makelinks -maxsize 1e6 -binsize 2e6 -numall 20 > links.txt
```

The link data file stores links as two lines, one line for the start of the link and one for the end.

```
hs1 46417 61698 mm2 111313326 111326293
hs1 395264 426112 mm1 18566118 18592077
...
```

In older versions of Circos, links were represented on two lines that were associated together using a unique identifier, which is the first field. The identifier can be any string, as long as it is unique for each link.

Lesson 3 – Coloring Links by Position

Rules are used to change the format of data in the figure based on position, value and other characteristics of the data. Because links are typically dense and span many regions (some regions may be interesting, and some not), rules greatly aid in identifying meaningful patterns in links and communicating the results to the reader.

In this lesson, we'll see how to color links based on their start and end positions. When comparing synteny across a large number of chromosomes, it is useful to identify the relationship between a given chromosome and all others of the other species. This can be done by coloring all links that start (or end) at a given chromosome.

Let's start by coloring all links by their mouse chromosome. To do this, we'll extract the "11" from the chromosome name `mm11` by removing the first two characters

```
substr(var(chr),2)
```

`substr()` extracts a portion of a string from an offset. Once we have this, we'll add prefix "`chr`" and add suffix "`_a4`".

```
sprintf("chr%s_a4",substr(var(chr),2))
```

In effect, the chromosome `mmN` is assigned a color `chrN_a4`. To tell Circos that this should be evaluated as code, wrap it in `eval()`.

```
<rule>
condition   = 1
color       = eval(sprintf("chr%s_a4",substr(var(chr2),2)))
...
</rule>
```

Link coloring can be restricted to a specific chromosome by changing the condition. The rule below will color all links that are on `mm11` (start or end at this chromosome) using color `chr11` (with transparency, `_a3`), change the `thickness` of the link to 2 pixels and set their `z` value to be higher than all other links, thus making these links drawn on top.

```
<rule>
condition   = on(mm11)
color       = chr11_a3
z           = 10
thickness   = 2p
</rule>
```

The result is shown in Figure 7.

Instead of `on()`, you can use `from()` and `to()` to test whether a link starts and ends at a given chromosome.

Rules can be combined into rule chains. Multiple rules can be designed and applied in sequence to each data set. By default, rules are applied in order of their appearance and the first rule that

matches terminates the rule chain. If you wish for a data point to continue to be tested with additional rules after being matched, use `flow=continue`.

Below is an extension of the rule above. The condition of the second rule requires that a data point have color `chr11_a3` (i.e. as assigned by the previous rule, thus this condition tests whether the previous rule matched) as well as requires that the start and end position be within 20-50Mb on hs1 (all links have their start assigned to hs1). When more than one condition is defined, all need to match. The rule colors these links red (with transparency, `_a3`), sets their `thickness` to 3 pixels and their `z` value to a high value, to make these links drawn on top (Figure 8).

```
<rule>
condition   = on(mm11)
color       = chr11_a3
z           = 10
thickness   = 2p
# links that pass are tested by remaining rules
flow        = continue
</rule>

<rule>
condition   = var(color) eq “chr11_a3”
condition   = var(start1) > 20e6
condition   = var(end1) < 50e6
color       = red_a3
z           = 20
thickness   = 3p
</rule>
```

We could have tested the second rule using

```
condition   = on(m11)
condition   = var(start1) > 20e6 && var(end1) < 50e6
```

and obtained the same result. But for the purpose of the lesson, the condition `var(color) eq “chr11_a3”` explicitly demonstrates that a rule’s condition can depend on a value set by a previous rule.

Lesson 4 – Bundling Links

As you can see from Figure 4, when a large number of links is drawn, the distribution of links can be difficult to discern. The links appear to cover the figure uniformly, making it hard to identify regions that are linked by multiple links. Moreover, using lines to depict links hides the size of the ends of the link.

Using the `bundlelinks` tool (see `tools/bundlelinks` in the Circos distribution) you can process a link file to generate a new file in which links have been bundled together, to form larger links, based on the distance between and size of the individual links.

http://www.circos.ca/documentation/tutorials/utilities/bundling_links

This process is illustrated in Figure 9. Links that connect the same chromosomes (e.g. chrA and chrB, or chrA and chrC) are eligible to be grouped into a bundle (Figure 9A). A bundle is seeded from a single link, with additional links added to the bundle if the ends of the link are within `max_gap` to the bundle (Figure 9B). A different cutoff can be imposed on the start and ends of the link (useful if you are comparing different genomes and wish to control density of links in a bundle at only one end of the bundle). Bundles can be filtered by imposing a limit on the minimum number of links in a bundle (Figure 10).

The script `3/data/create.tracks` includes calls to `bundlelinks` to generate the bundle tracks for this session. For example, to bundle all links that are within 3Mb on hs1 into bundles with at least 3 links, the `max_gap_1` parameter is used to limit the distance between link starts.

```
> bundlelinks -links links.txt -max_gap_1 3e6
              -min_bundle_membership 3  > bundles.txt
```

The bundled links are shown in Figure 11.

```
<links>
<link>
ribbon        = yes
file          = ../data/bundles.txt
bezier_radius = 0r
radius        = 0.85r
thickness     = 0p
color         = grey_a5

</link>
```

First, notice that `ribbon=yes` is set. This turns the links into ribbons—curved regions whose ends reflect the size of the ends of the link in the data file. Because bundles have ends that are typically much larger than the ends of the links from which they are formed (see Figure 9A), using ribbons to draw bundles is a good idea.

```
<rule>
condition   = 1
color       = eval(sprintf("chr%s_a1",substr(var(chr2),2)))
radius2     = 0.99r
z           = eval(var(size2))
```

```
flow        = stop
</rule>
```

From this lesson onwards, we'll draw the link ends closer to the mouse chromosomes. This can be done by adjusting the `radius2` parameter, which controls the radial position of a link's end (`radius1` controls position of the link's start).

Finally, the `z` value is made proportional to the size of the link on hs1. This has the result of larger bundles drawn on top.

If you would like to change the order of the links to draw the small links on top, change the `z` value to be inversely proportional to the link size. It's enough to make the `z` value to be the negative of the link size. Since links are drawn in order of ascending z, this scheme will place small links last.

```
z = eval(-var(size1))
```

This change in `z` is shown in Figure 12.

Let's focus the figure on links from `mm5`, `mm8` and `mm11`. The first rule will change the color of all links grey using the 9-color greys Brewer palette. The second rule will trigger only for links on chromosomes mm5, mm8 or mm11 and change the color of these links to `chr5`, `chr8` and `chr11`.

```
<rules>
<rule>
condition   = 1
color       = eval(sprintf("greys-9-seq-%d_a5",remap_round(var(size1),0,1e6,2,9)))
z           = eval(-var(size2))
flow        = continue
</rule>

<rule>
condition   = on(mm(5|8|11))
color       = eval(sprintf("chr%s_a1",substr(var(chr2),2)))
z           = 10
</rule>
</rules>
```

The effect of these two rules is shown in Figure 13.

WHAT IS SHOWN IN THE FIGURE?

By bundling the links we've created a visualization of syntenic blocks between human chromosome 1 and the mouse genome. Evaluating these blocks is made difficult when a large number of links is present in the data set. Drawing all the links (Figure 4) results in a busy figure which is difficult to interpret.

By requiring that links be turned into bundles, which are formed from at least 3 links that are no further than 3Mb apart (pairwise) on hs1, it is possible to show patterns of similarity which involve multiple links. Recall that the links shown in Figure 4 correspond to individual

alignments from the Chain/Net UCSC track. By bundling these alignments together, we are effectively creating a large gapped alignment.

The `3/data/create.tracks` a script creates six different bundle sets. Each set requires that bundled links be no further than 3Mb apart, but varies the minimum number of links per bundle from 2 to 20. These six bundle sets are shown in Figure 10.

```
# sessions/3/data/create.tracks
...
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 20 > bundles.0.txt
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 10 > bundles.1.txt
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 5  > bundles.2.txt
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 4  > bundles.3.txt
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 3  > bundles.4.txt
bundlelinks -links links.txt -max_gap_1 3e6
            -min_bundle_membership 2  > bundles.5.txt
...
```

Lesson 5 – Histograms

We've seen histograms in the previous session. These are histograms whose content is generated from a link file using binlinks. Their data summarizes the number (or size) of links that start or end within a certain position window.

In Figure 14 three histograms are shown, defined by the blocks below.

The first block defines the histogram from `../data/histogram.mm.txt`, which is the black histogram found outside the mouse ideograms. This histogram shows the total size of links fall within 5Mb windows on the mouse chromosomes.

The second block, from `../data/histogram.mm.bundles.txt`, counts the total size of bundles, created in the previous lesson, per 5Mb on mouse chromosomes. This histogram is red, without a fill. Notice that the scale for this histogram is 0-35Mb, in contrast to the link density histogram whose scale is 0-50kb.

Finally, the third block, from `../data/histogram.hs.stacked.txt`, shows the contribution of links from each mouse chromosome to every 1Mb window of hs1. A stacked histogram takes a list for its fill_color parameter. In this case, the list is set to '`chrs`' which is the list of colors chr1...chr22,chrX,chrY.

```
...
hs1 81000000 81999999 1.0000,0.0000,0.0000,1.0000,0.0000,0.0000,
                      1.0000,0.0000,1.0000,0.0000,0.0000,0.0000,
                      0.0000,0.0000,1.0000,2.0000,0.0000,0.0000,0.0000
...
```

```
<plot>
type       = histogram
min        = 0
max        = 50e3
r0         = 1r
r1         = 1r+43p
file       = ../data/histogram.mm.txt
fill_color = black
thickness  = 0
</plot>

<plot>
type       = histogram
min        = 0
max        = 35e6
r0         = 1r
r1         = 1r+43p
file       = ../data/histogram.mm.bundles.txt
thickness  = 2
color      = reds-5-seq-4
z          = 10
</plot>
```

```
<plot>
type       = histogram
min        = 0
max        = 25
r0         = 1r
r1         = 1r+43p
file       = ../data/histogram.hs.stacked.txt
fill_color = chrs
</plot>
```

The `binlinks` tool can create normalized stacked histograms, in which the total of each bin is normalized to 1. The maximum for the stacked normalized histogram is set to 1. This version of the stacked histogram is shown in Figure 15.

```
<plot>
show       = yes
type       = histogram
min        = 0
max        = 1
r0         = 1r
r1         = 1r+43p
file       = ../data/histogram.hs.stacked.norm.txt
fill_color = chrs
#sort_bin_values  = yes
</plot>
```

Finally, if `sort_bin_values=yes`, the order of bins is based on value, placing larger bins at the bottom of the stacked histogram (Figure 16). This approach can create a busy track, since the order for each bin varies based on individual values, but it's easy to see which bin makes the largest contribution.

WHAT IS SHOWN IN THE FIGURE

Density histograms reflect the total number and size of links within a genomic window. These tracks were created from the link data file (`links.txt`) using the `binlinks` tool.

```
binlinks -links links.txt -link_end 1 -output_style 0 -bin 5e6
  > histogram.mm.txt
binlinks -links links.txt -link_end 0 -output_style 1 -bin 1e6 | sed 's/mm/chr/'
  > histogram.hs.txt
binlinks -links links.txt -link_end 0 -output_style 3 -bin 1e6 -num
  > histogram.hs.stacked.txt
binlinks -links links.txt -link_end 0 -output_style 3 -bin 1e6 -num -normalize
  > histogram.hs.stacked.norm.txt
```

The `-link_end` controls which end of the link is binned (`0` start, `1` end, `2` both) and the `-output_style` controls the detail in the histogram file (apply color, create stacked histogram, etc). When `-num` is used, the number (rather than size) of links is binned and `-normalize` applies a normalization factor to make the total bin size of stacked histograms unity.

For more information about `binlinks`, see

http://www.circos.ca/documentation/tutorials/utilities/density_tracks/

Lesson 6 – Scatter Plots

Scatter plots use the same input format as histograms, but draw the data using glyphs. Glyph size and color can be controlled, as can glyph shape. You can probably guess that we'll be using rules to adjust glyphs!

In Figure 17, the scatter plot shows the average conservation score between human and mouse for each 1Mb window on hs1.

The default appearance of each glyph is a small grey square, as defined by this block.

```
<plot>
type       = scatter
file       = ../data/scatter.cons.txt
min        = 0.39
max        = 0.55
r0         = 0.80r
r1         = 0.90r
glyph      = square
glyph_size = 3
fill_color = grey
...
```

You'll notice what may appear to be a strange first block in the rule chain for this plot.

```
<rule>
flow = stop
</rule>
```

This is a special kind of rule that allows the short-circuiting of all other rules. There is no explicit condition—`condition=1` is assumed. This rule is equivalent to the following

```
<rule>
condition = 1
</rule>
```

Rules adjust the color of the glyph based on the value. Independently, I calculated the statistics for the scatter plot, which are

$n = 229$

mean = 0.455

10% percentile = 0.416

90% percentile = 0.495

The rules below successively test each data point and color large values green, small values red and values very close to the average a dark grey.

```
<rule>
```

```
# 90% percentile
condition  = var(value) >= 0.489
color      = green
flow       = continue
</rule>

<rule>
# 10% percentile
condition  = var(value) <= 0.416
color      = red
flow       = continue
</rule>

<rule>
# within 1 std of mean
condition  = abs(var(value) - 0.455) < 0.01
color      = dgrey
flow       = continue
</rule>

<rule>
condition  = 1
glyph_size = eval(remap_round(abs(var(value) - 0.455),0,0.1,5,20))
z          = eval(remap_round(abs(var(value) - 0.455),0,0.1,1,100))
glyph      = circle
flow       = continue
</rule>
```

The size of the glyph can be adjusted – and be made a function of the data value – by the last rule. The size is made proportional to the deviation, and mapped onto the range [5,20]. The z-value is made larger for points further from the average in the same way. The plot with these rules in action is shown in Figure 18.

The scatter plot can be subverted into a type of glyph track, forcing the value of each point to the baseline, using a rule like

```
<rule>
condition  = 1
value      = 0.47 # midway between min and max
</rule>
```

Note that we had to change the glyph size and value in two separate rules, because the rule that changes value might adversely affect the glyph size. By setting the value of each data point to the midpoint between `min` and `max`, all the glyphs are at the baseline as shown in Figure 19.

There are other ways of achieving this, such as setting `r0` and `r1` to be the same value (thus collapsing a track onto a single radial position), or setting a very small min and very large max (e.g. `min = -100`, `max = 100`).

Lesson 7 – Track Automation — Part 1

If you have a large number of input files for similarly formatted tracks (e.g. 20 heatmaps), you can use track counters to automate the creation and placement of tracks.

http://circos.ca/documentation/tutorials/recipes/automating_tracks/

http://circos.ca/documentation/tutorials/recipes/automating_heatmaps/

Counters are an advanced automation topic, so I've left them until the end of this session. Take a look at the heatmap blocks below. The first one initializes a counter named h with value 1. The value of this counter is available via `counter(h)`. Subsequent blocks increment the value of the counter before the block is processed using `pre_increment_counter`.

```
h0 = 0.75 # heatmap start
hs = 0.07 # heatmap step
hw = 0.06 # heatmap width

<plots>

<plot>
init_counter = h:1
type  = heatmap
file  = ../data/heatmap.counter(h).txt
r1    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs) + conf(hw) ))
r0    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs)))
color = eval(sprintf("chr%s_a3",counter(h)))
</plot>

<plot>
pre_increment_counter = h:1
type  = heatmap
file  = ../data/heatmap.counter(h).txt
r1    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs) + conf(hw) ))
r0    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs)))
color = eval(sprintf("chr%s_a3",counter(h)))
</plot>

<plot>
pre_increment_counter = h:1
type  = heatmap
file  = ../data/heatmap.counter(h).txt
r1    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs) + conf(hw) ))
r0    = eval(sprintf("%fr",conf(h0) + (counter(h)-1) * conf(hs)))
color = eval(sprintf("chr%s_a3",counter(h)))
</plot>

</plots>
```

The position (`r0`, `r1`) of each track is defined to be a function of the counter and the global variables that set the start (`h0`), width (`hw`) and spacing (`hs`) of each track.

For example, the parameters of the first track are evaluated with `counter(h)=1`

```
file  = ../data/heatmap.1.txt
r0    = 0.75r # 0.75 + (1-1) * 0.07
r1    = 0.81r # 0.75 + (1-1) * 0.07 + 0.06
color = chr1_a3
```

The result will be three heatmaps (Figure 20), each sourced from a different file (heatmap.1.txt, heatmap.2.txt, heatmap.3.txt), each at different r0/r1 and using a different color. I emphasize that the block definition for each heat map is *identical* (`pre_increment_counter` acts the same as `init_counter` when called on an undefined counter name). The difference between the blocks is due to the counter changing values.

It is just as easy now to create 19 such heatmaps, one for each mouse chromosome. First, we save the heatmap block in an external configuration file in this lesson's `etc/` directory.

By including this file 19 times, you make 19 heatmaps with very little work. This is shown in the next lesson.

Lesson 8 – Track Automation — Part 2

A very compact configuration file for this lesson's image in Figure 21, which shows 19 heatmaps.

```
heatmap_start = 0.75
heatmap_step  = 0.01
heatmap_width = 0.01

<plots>

<<include ../5/etc/histograms.conf>>
<<include ../6/etc/scatter.conf>>

<<include ../7/etc/heatmap.conf>> # mm1
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>

<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>

<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>

<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>> # mm19

</plots>
```

If you have a very large number of tracks to include, it can be useful to define a file that includes a number of heatmaps (e.g. 6) and then include this file several times, thereby including 6 heatmaps at a time.

```
# etc/heatmaps.6.conf
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
<<include ../7/etc/heatmap.conf>>
```

and now to create 19 heatmaps,

```
<<include ../7/etc/heatmap.6.conf>>
<<include ../7/etc/heatmap.6.conf>>
<<include ../7/etc/heatmap.6.conf>>
<<include ../7/etc/heatmap.conf>>
```

The last include statement includes a single heatmap to make up `19 = 3*6 + 1`.

The final figure (Figure 22) shows how the counter index can affect how data is displayed.

```
# color tile by counter, alternating between black and red
color = eval(("black","red")[counter(h)%2])
```

◦

Figures

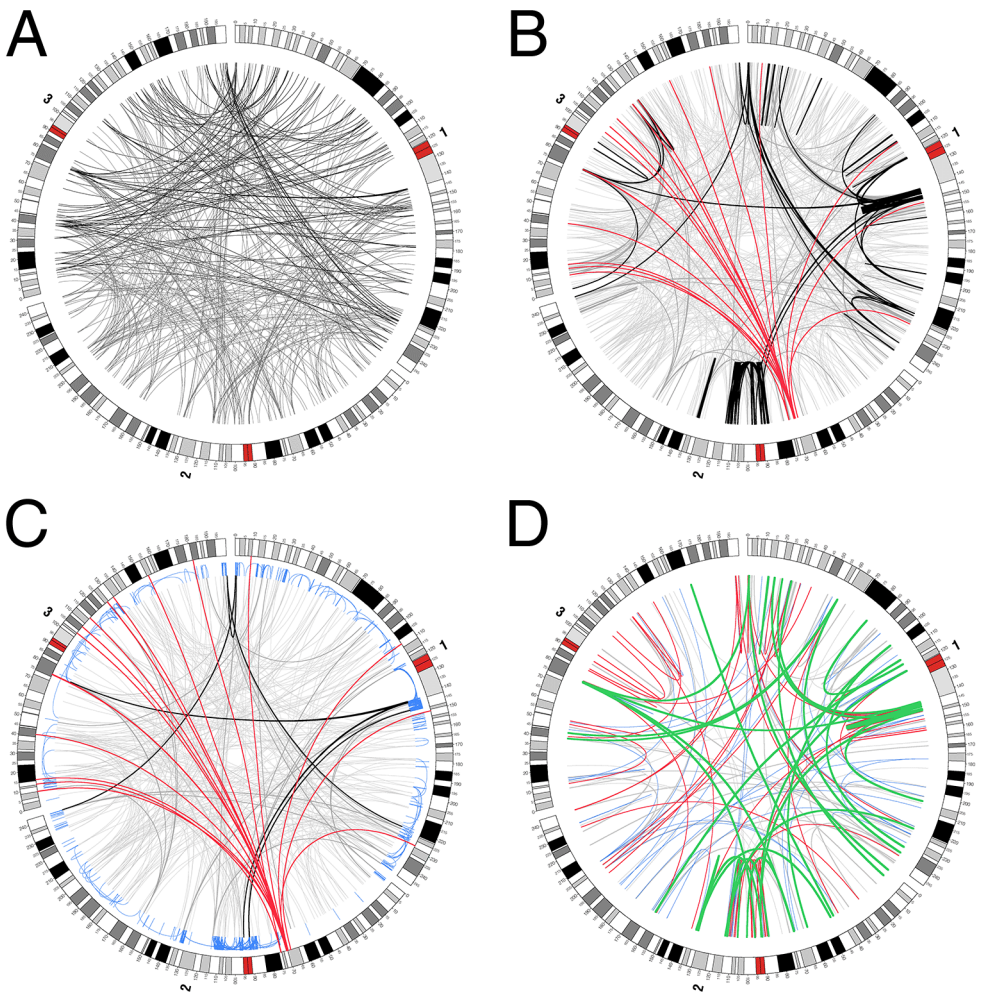

FIGURE 1

Examples of how rules and links are combined. (A) Original data set. (B) Color of certain links is modified using rules. (C) Geometry of nearby intra-chromosomal links has been adjusted to point the link outwards. (D) Rules were used to change the thickness of links.

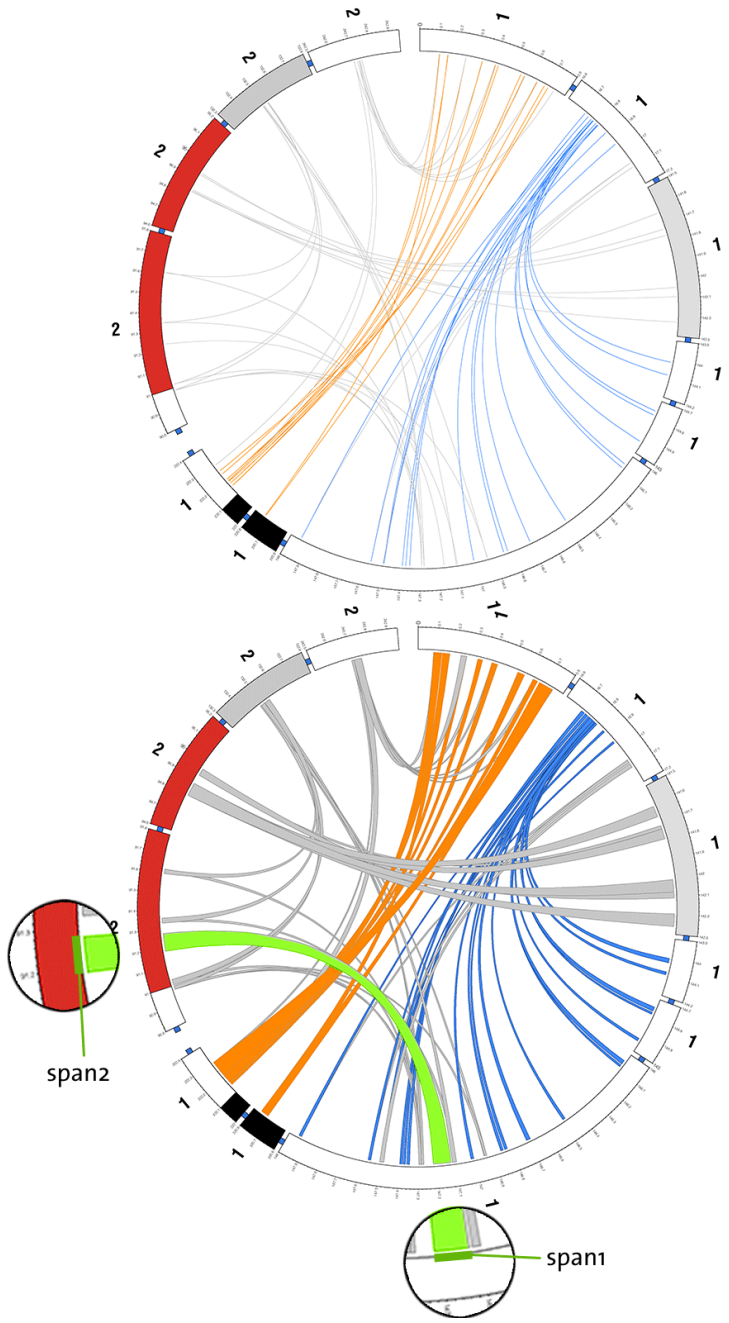

FIGURE 2

Links are defined by two regions, which can be of any size. *top* By default, links are drawn as lines (with adjustable, but constant, thickness). The lines start and end in the middle of the regions that define the link. *bottom* When the regions that define the link are large, it is helpful to use the thickness of the link to reflect the region size. To do this, links can be drawn as ribbons whose ends take on the thickness of the regions that define the link. When links are drawn as ribbons, thickness is not necessarily constant across the link. Depending on the orientation of the start and end regions, and the relative orientation of the scales of the ideograms that the link connects, ribbons can twist. This twisting can be explicitly controlled (e.g. all ribbons can be made flat, regardless of orientation of scale and link regions).

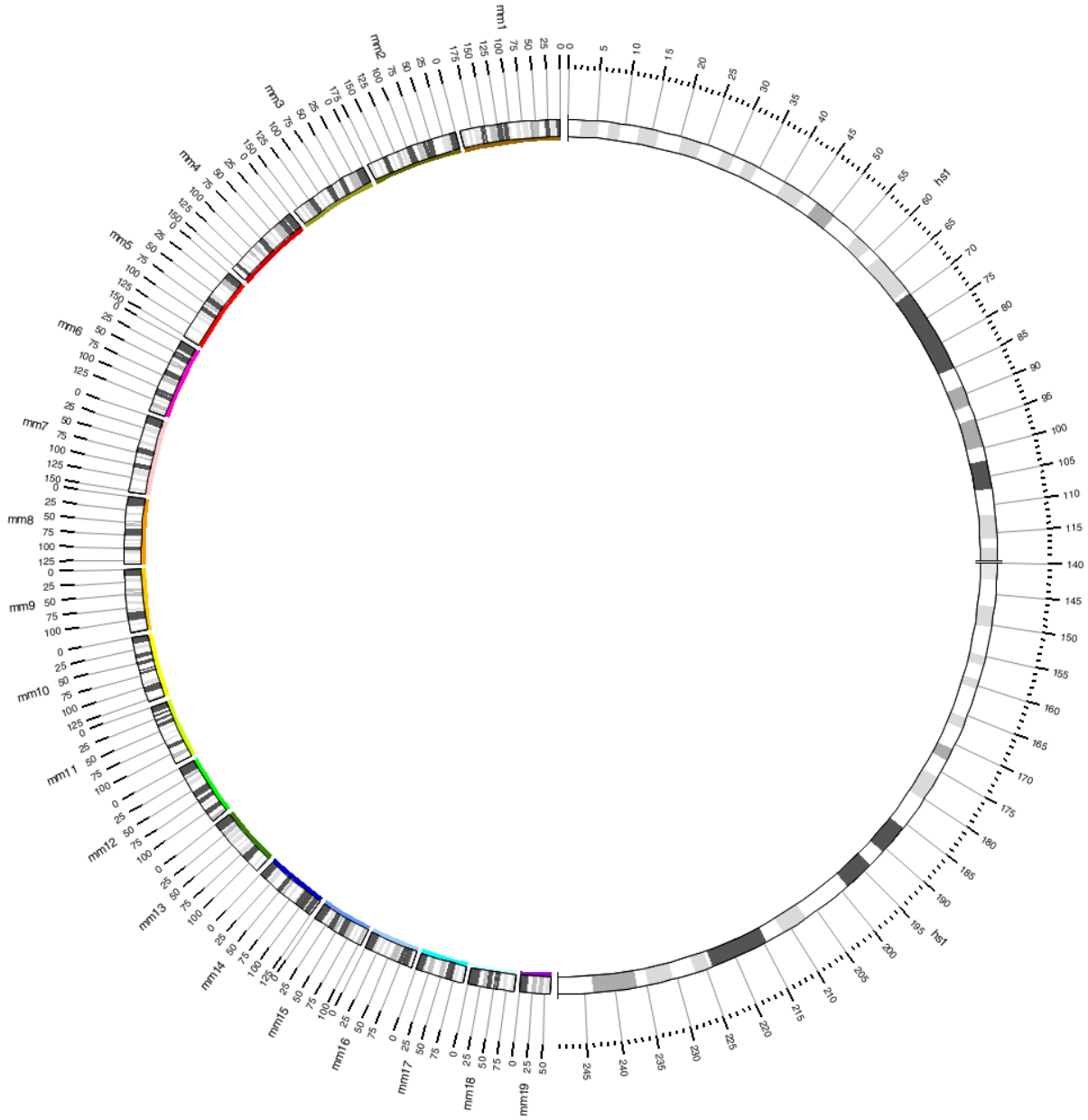

FIGURE 3

Mouse chromosomes 1-19 occupy ½ of the figure and human chromosome 1 is shown in the other ½.

The human chromosome has an axis break at 120-140Mb to remove the centromere from the display (there is no data for this region).

Mouse chromosomes are oriented counter-clockwise.

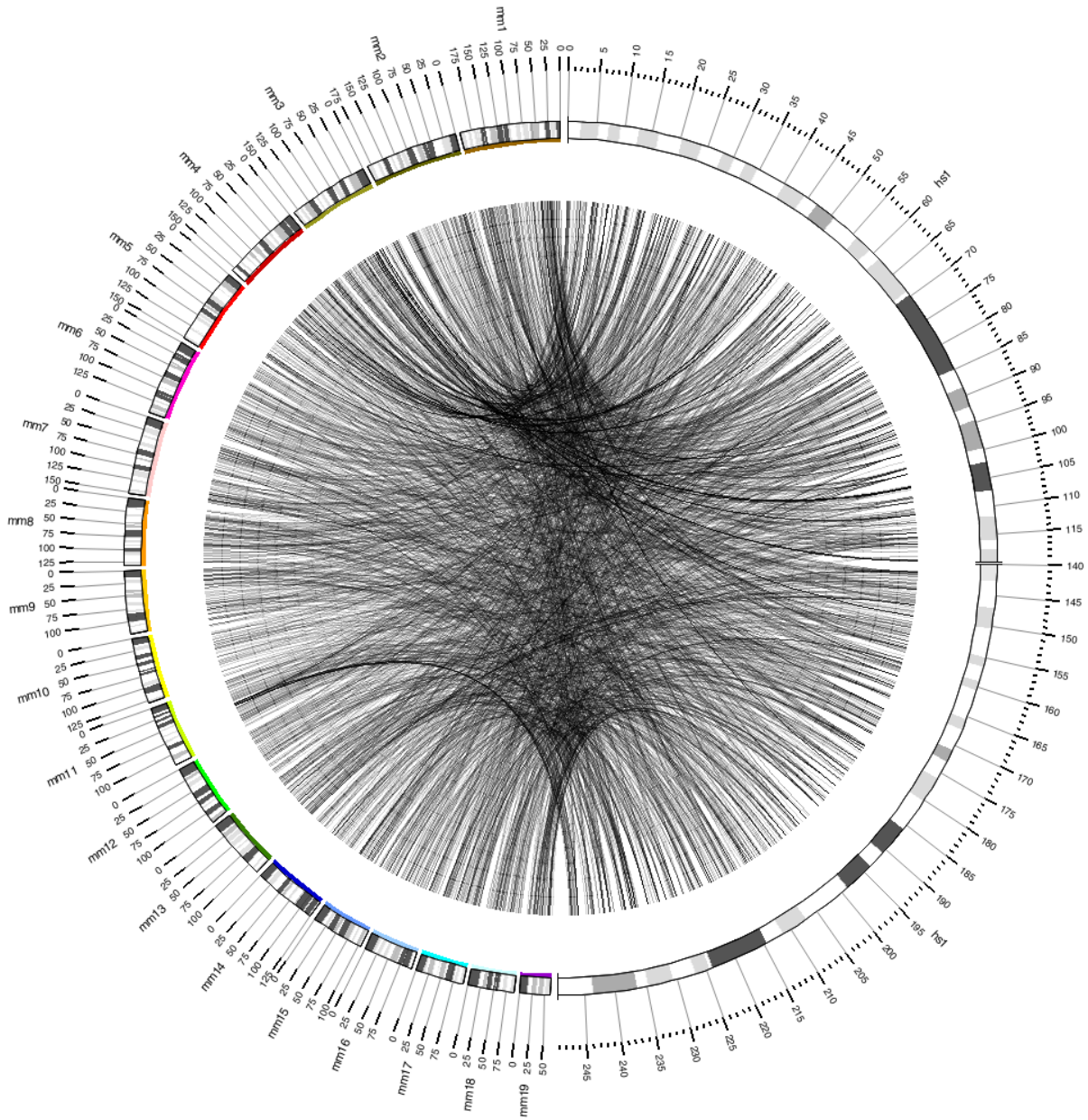

FIGURE 4

The links show 2,300 top alignments between human chromosome 1 and mouse chromosomes 1-19.

When transparency is used for link lines, it is possible to discern regions where the links are denser. The color for each link line here is `black_a5`. Compare this figure with Figure 5, where transparency was not used.

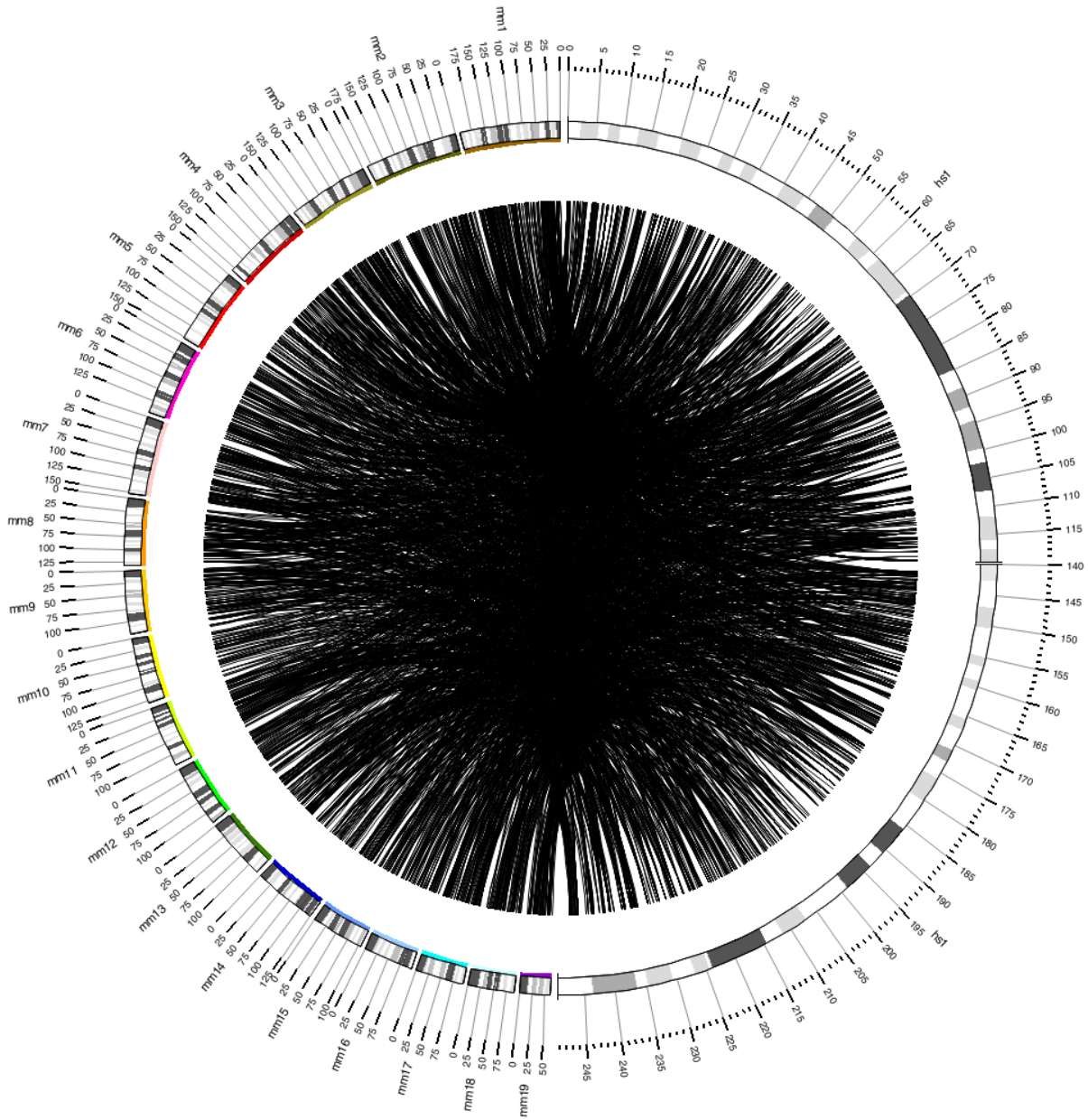

FIGURE 5

The links show 2,300 top alignments between human chromosome 1 and mouse chromosomes 1-19.

When transparency is not used for link lines, dense links form a solid shape making it impossible to discern regions where the links are denser. The color for each link line here is `black` (note, no `_aN` suffix). Compare this figure with Figure 4, where transparency was used.

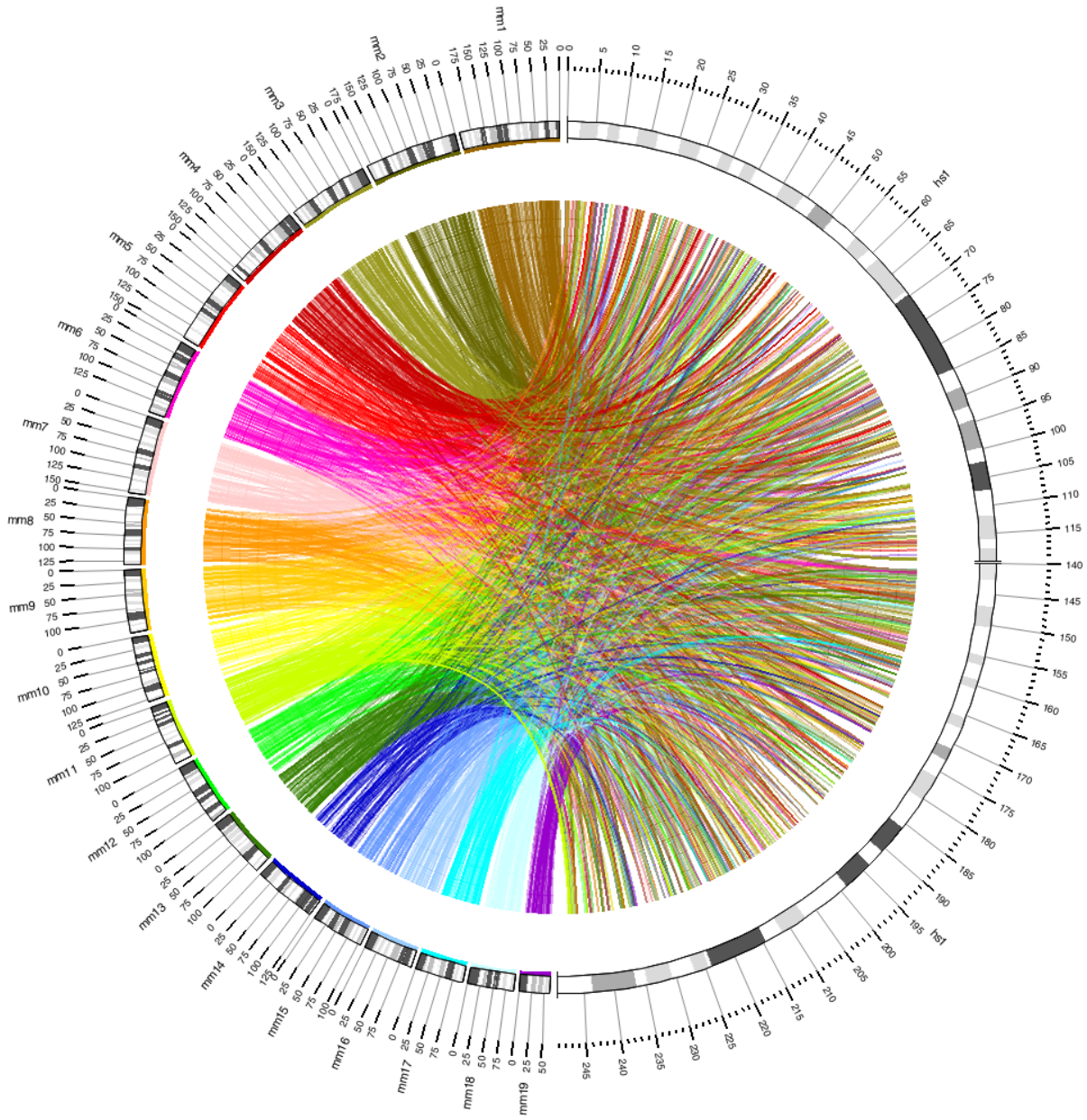

FIGURE 6

Each link is colored by the mouse chromosome from which it originates.

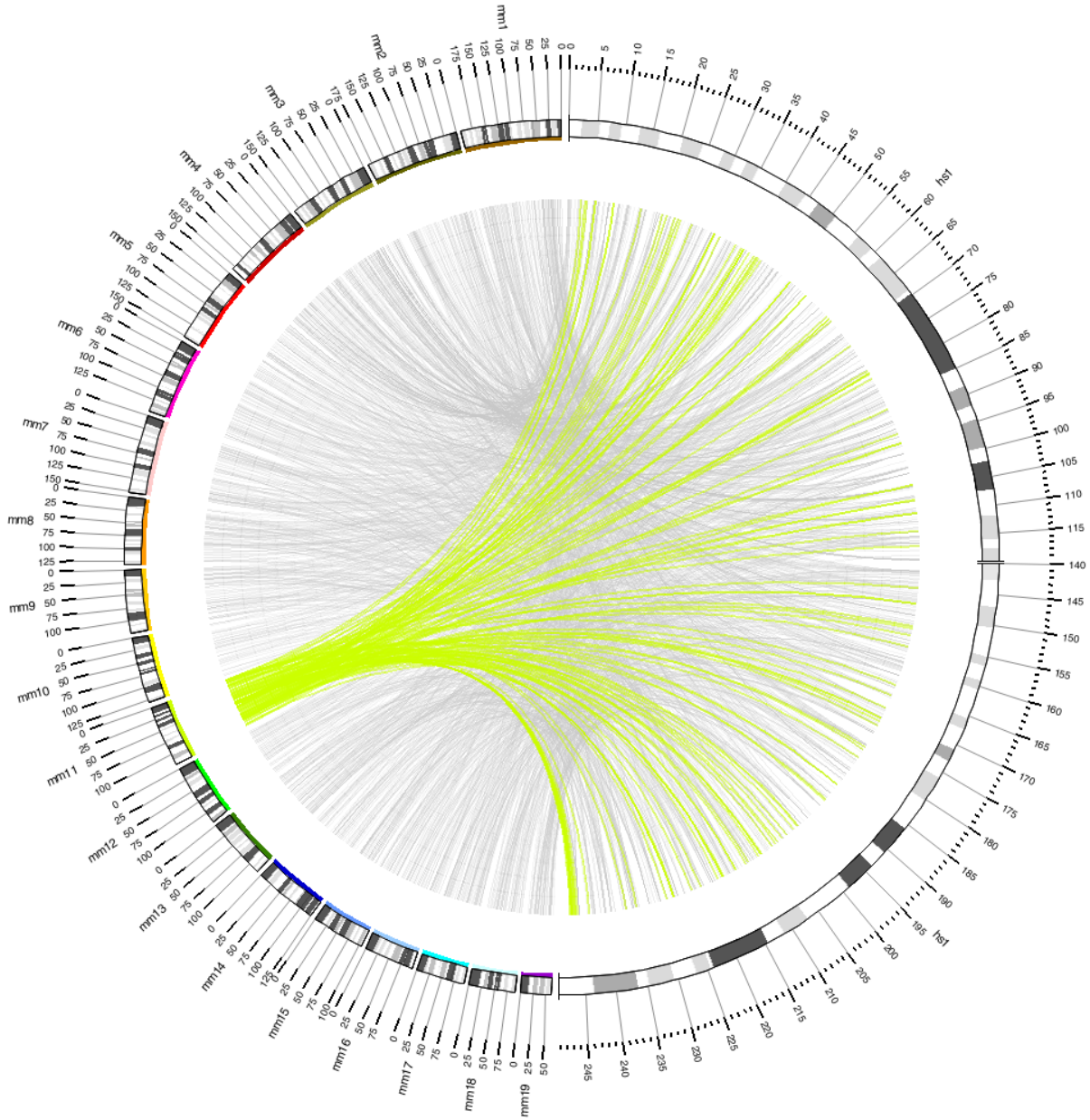

FIGURE 7

Rules are used to color all links that impinge on mouse chromosome 11.

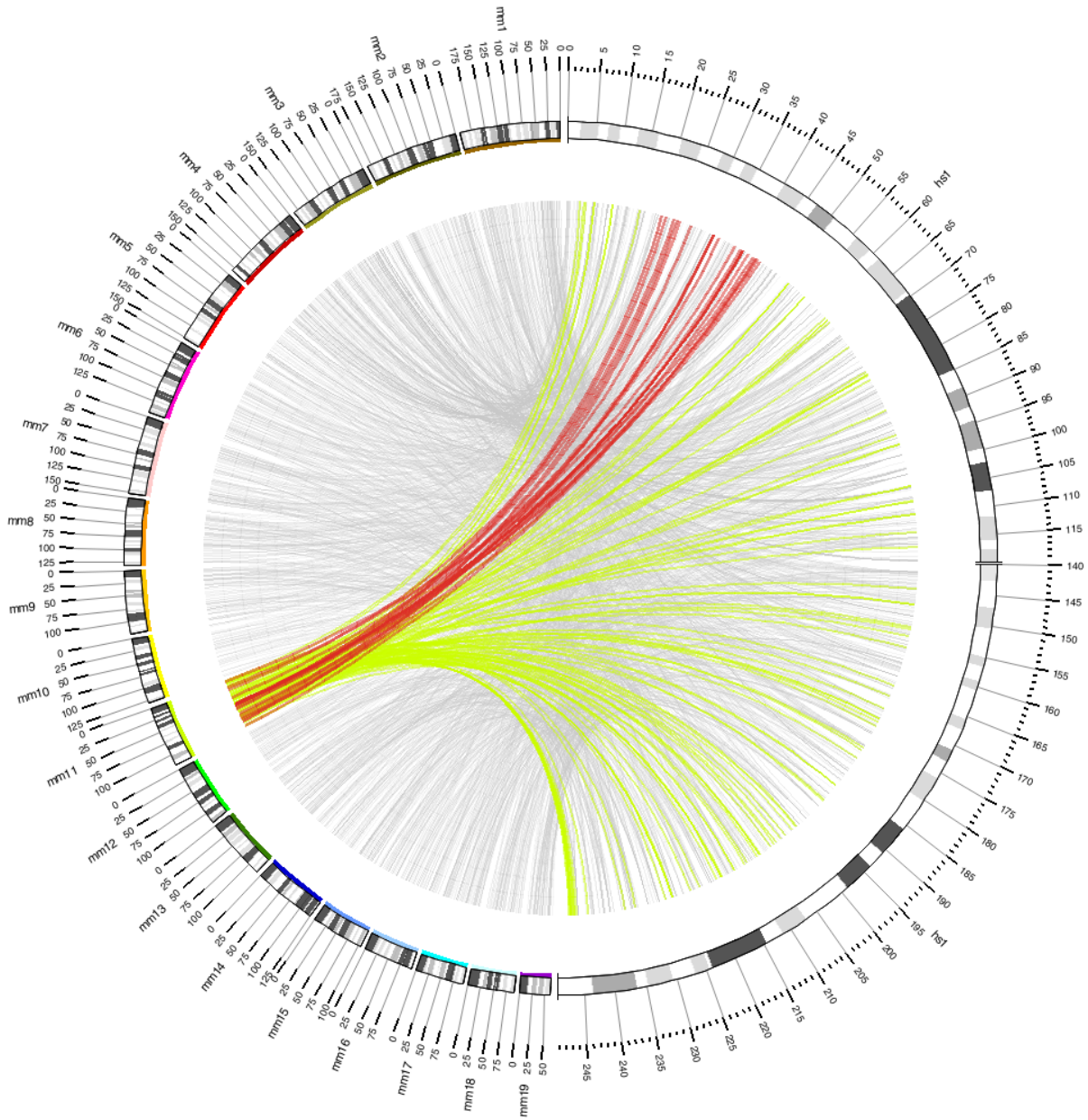

FIGURE 8

A second rule is added to uniquely color all mm11 links that start at 20-50Mb of hs1.

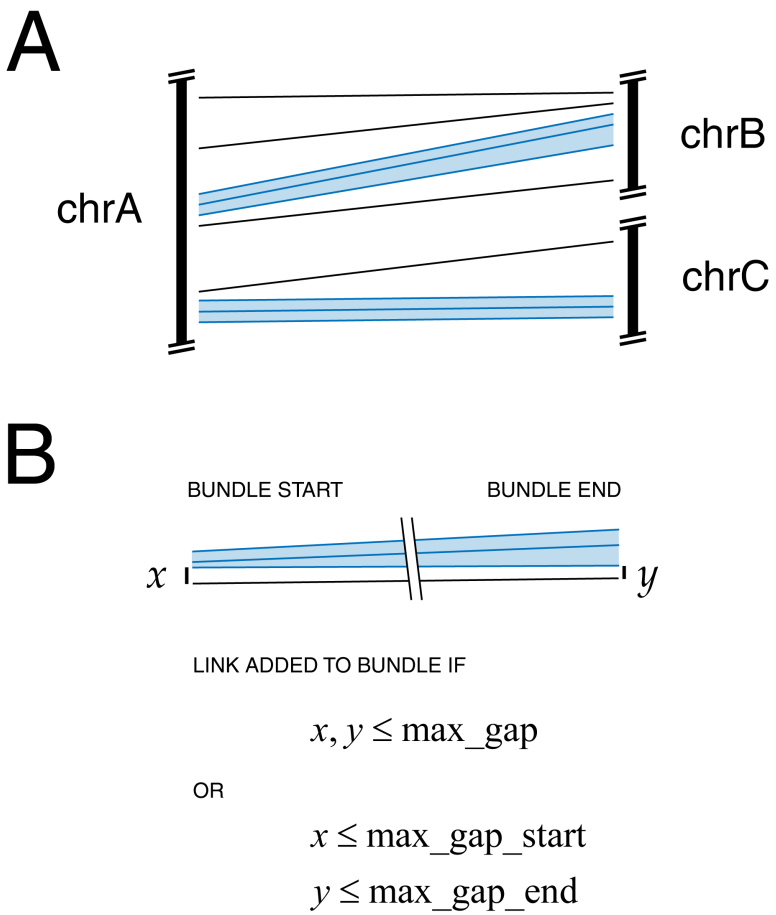

FIGURE 9

The `bundlelinks` tools is used to logically group adjacent links together, forming larger links. Links are bundled based on their size and distance to each other.

Bundles are best drawn as ribbons, rather than lines, because bundle ends typically span a significant section of an ideogram.

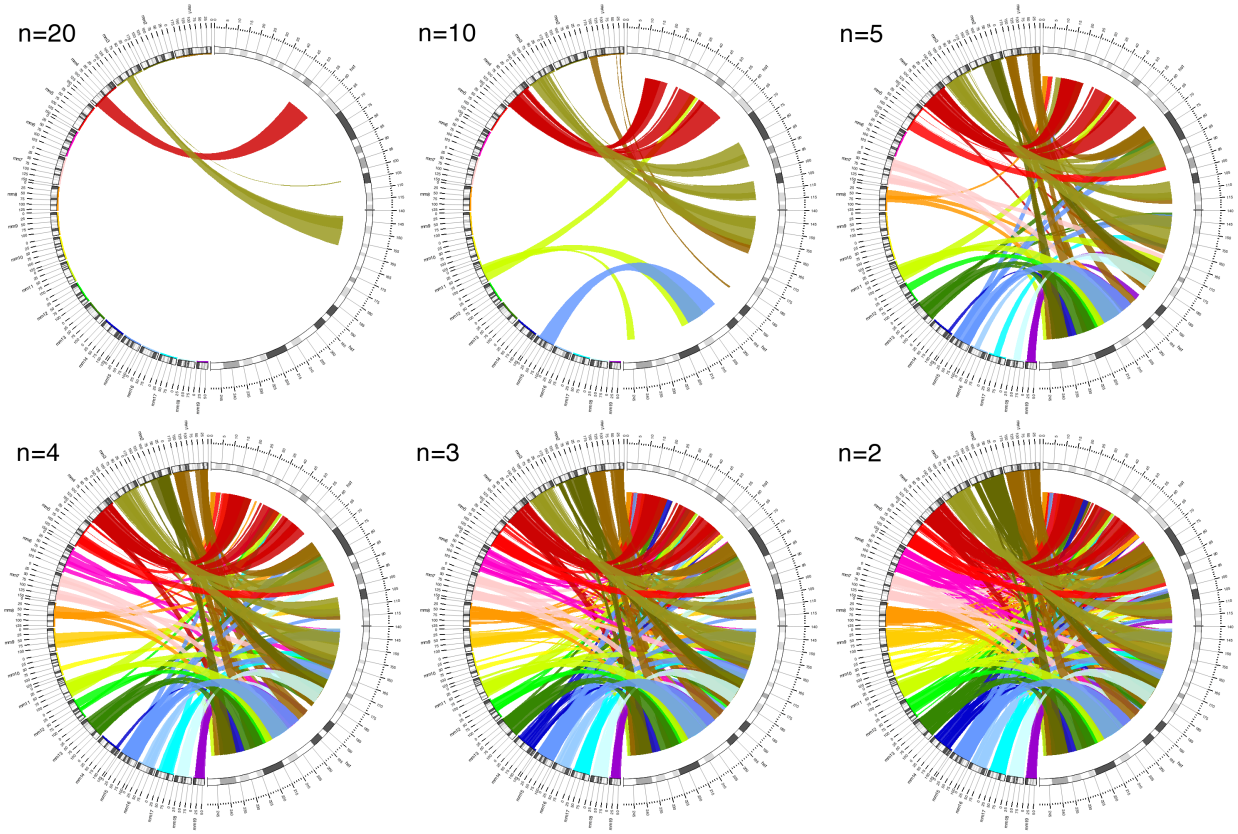

FIGURE 10

Varying the minimum number of links per bundle changes the sensitivity of bundling.

When a large number of links is required (e.g. n=20), only those regions that are connected by a large number of links are turned into bundles. When this number is decreased (e.g. `n = 10, n = 5, ...`), the number of bundles increases. If the cutoff is small (e.g. `n = 2, 3`), it is possible to create a large number of bundles, because fewer links are required to form a bundle.

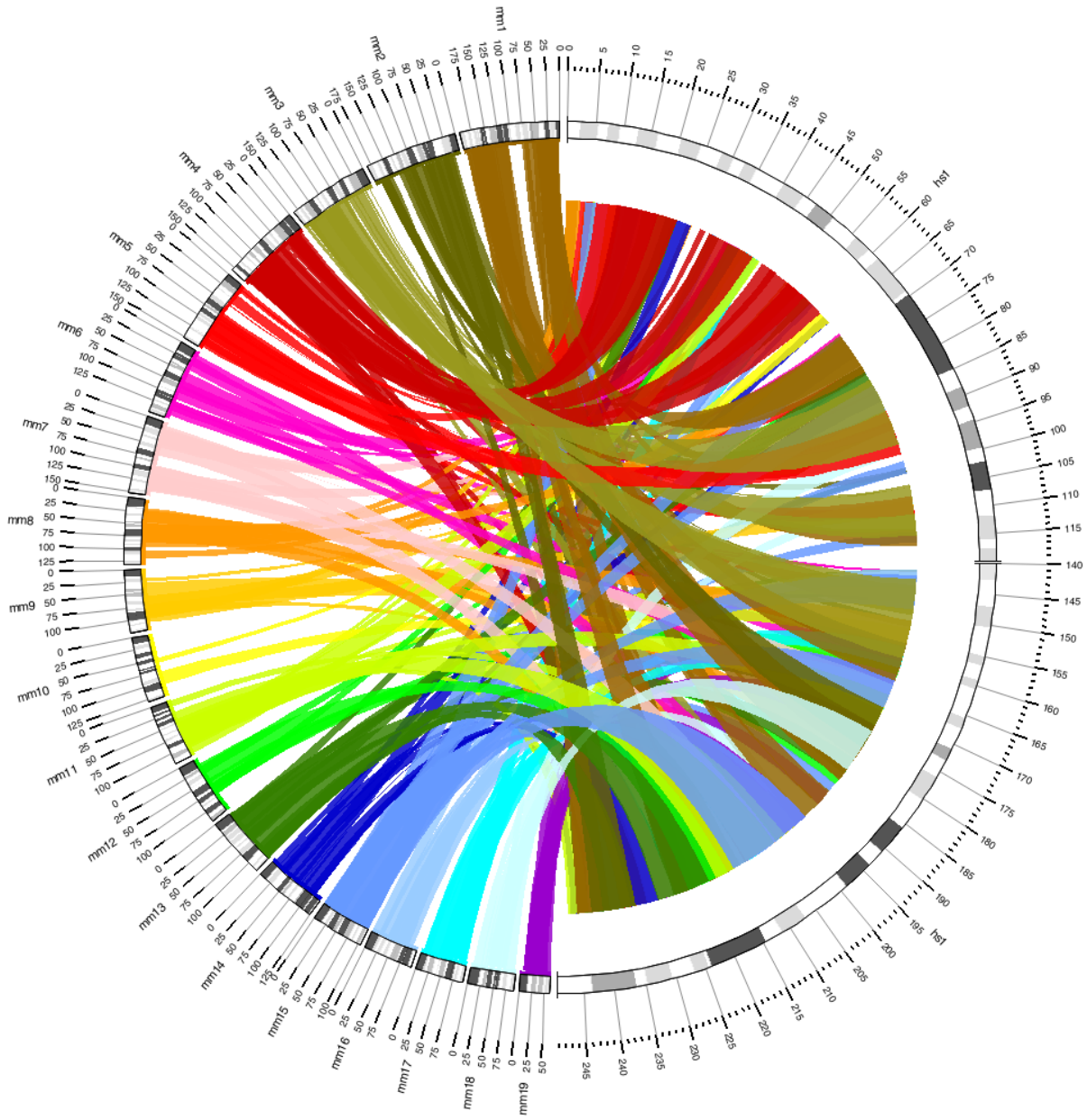

FIGURE 11

The result of bundling links shown in Figure 4. Using `radius2`, the ends of the links are drawn closer to the mouse chromosomes.

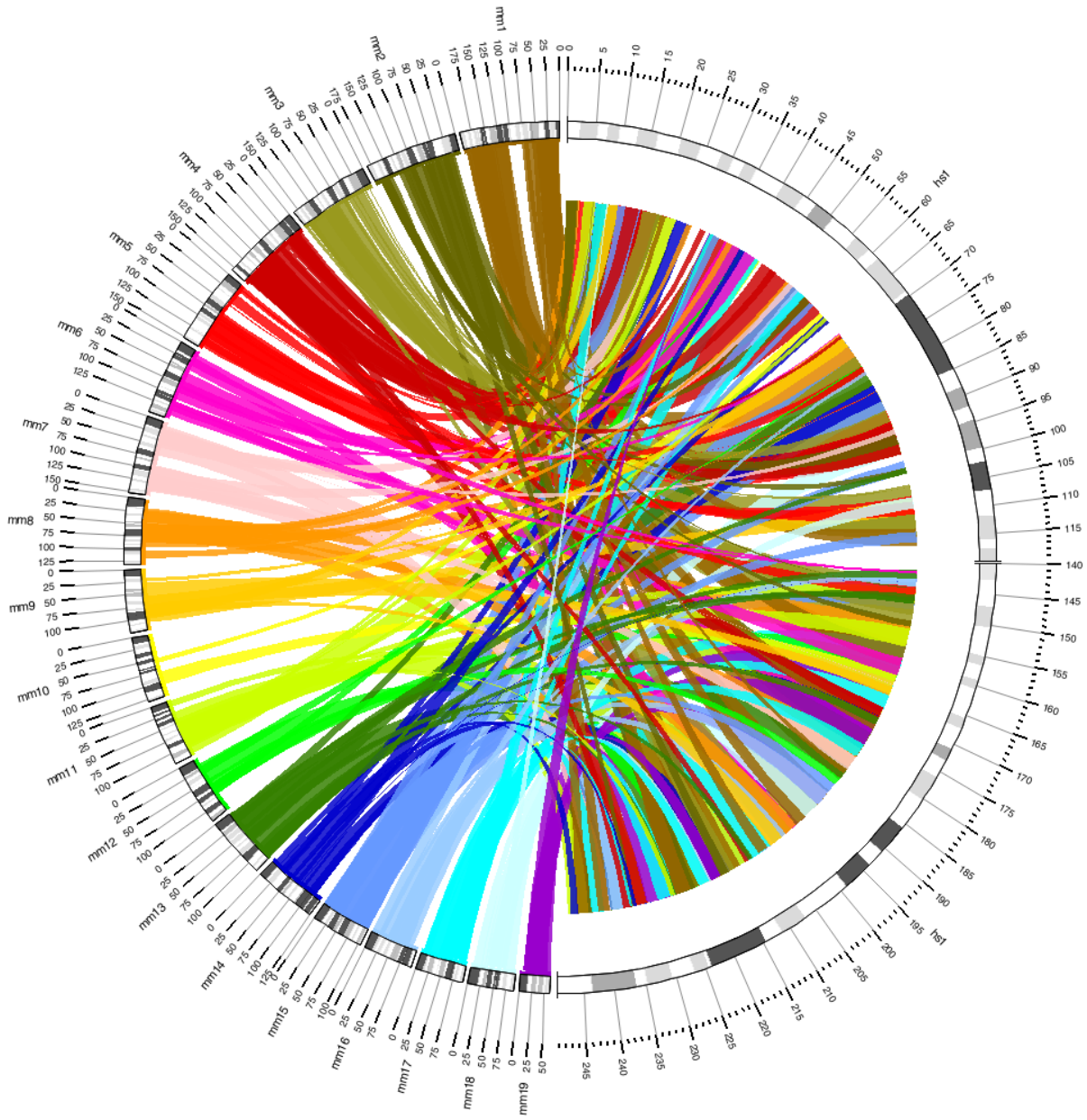

FIGURE 12

By setting the z value to be inversely proportional to link size, small links are drawn on top.

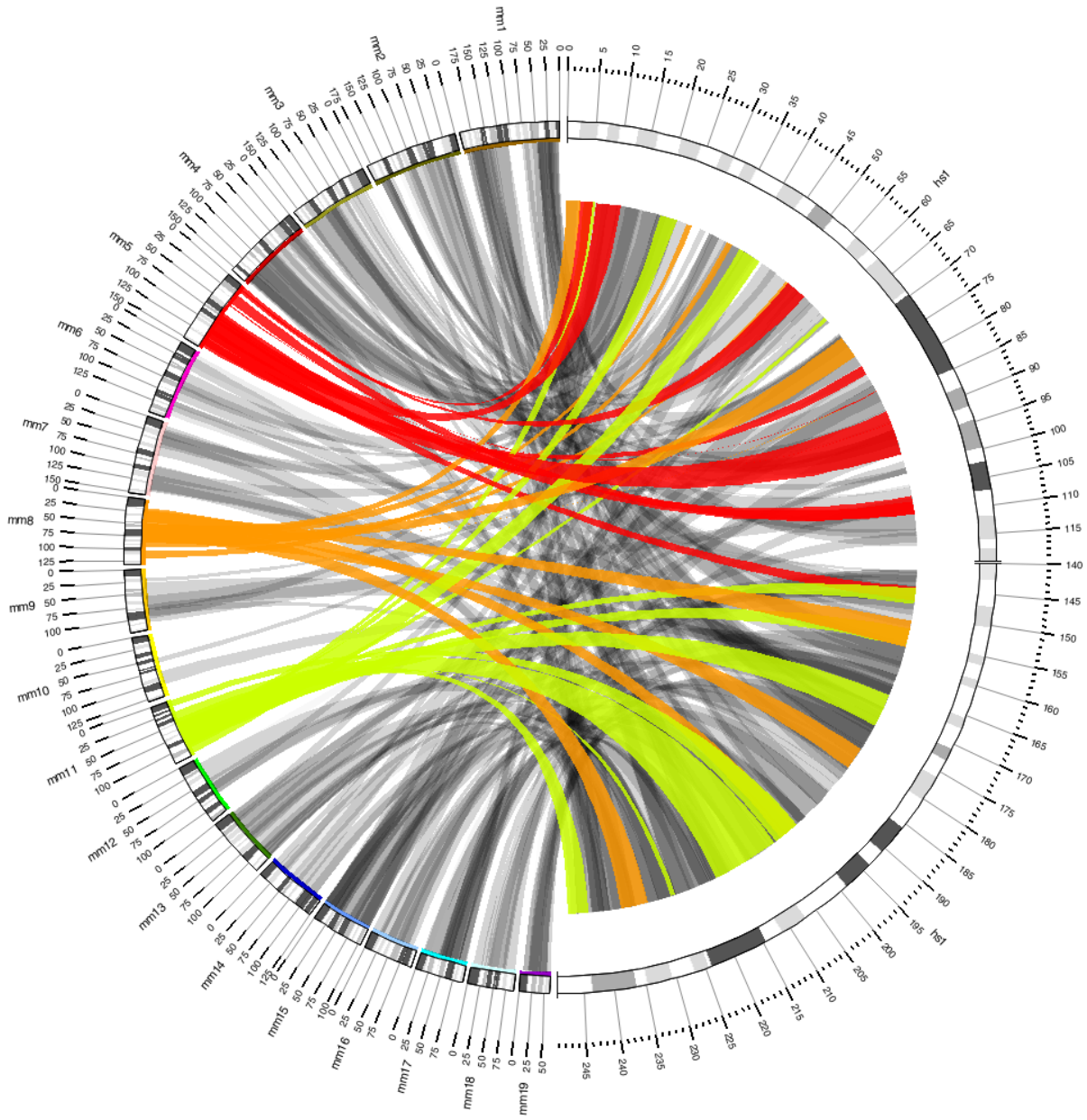

FIGURE 13

Links on mouse chromosomes 5, 8 and 11 are colored and the remaining links are made grey, in proportion to the size of the link.

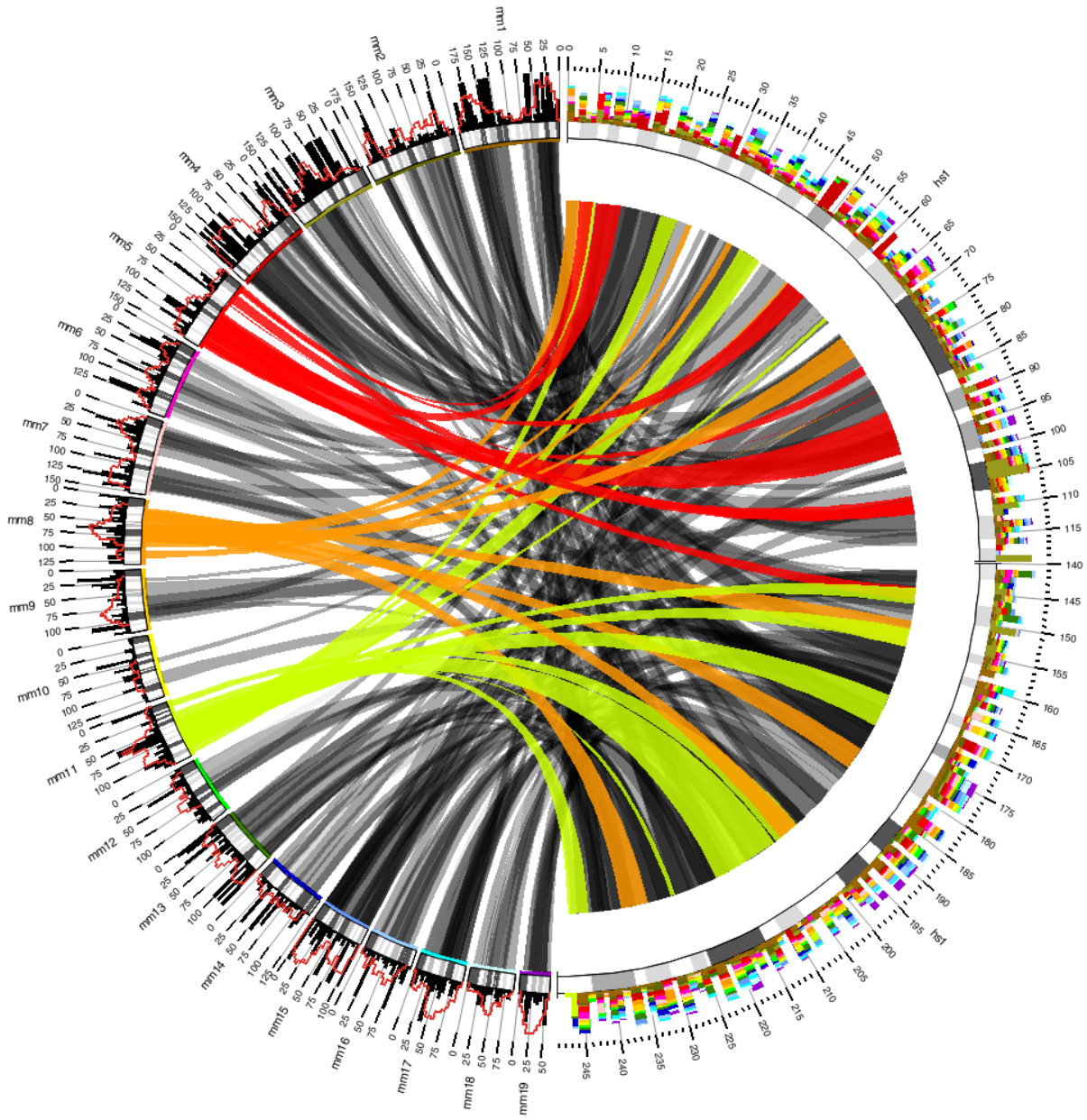

FIGURE 14

Three density histograms summarize information about the synteny between human chromosome 1 and the mouse genome.

The links in this figure are drawn as bundles, but the density histograms are calculated based on the individual links.

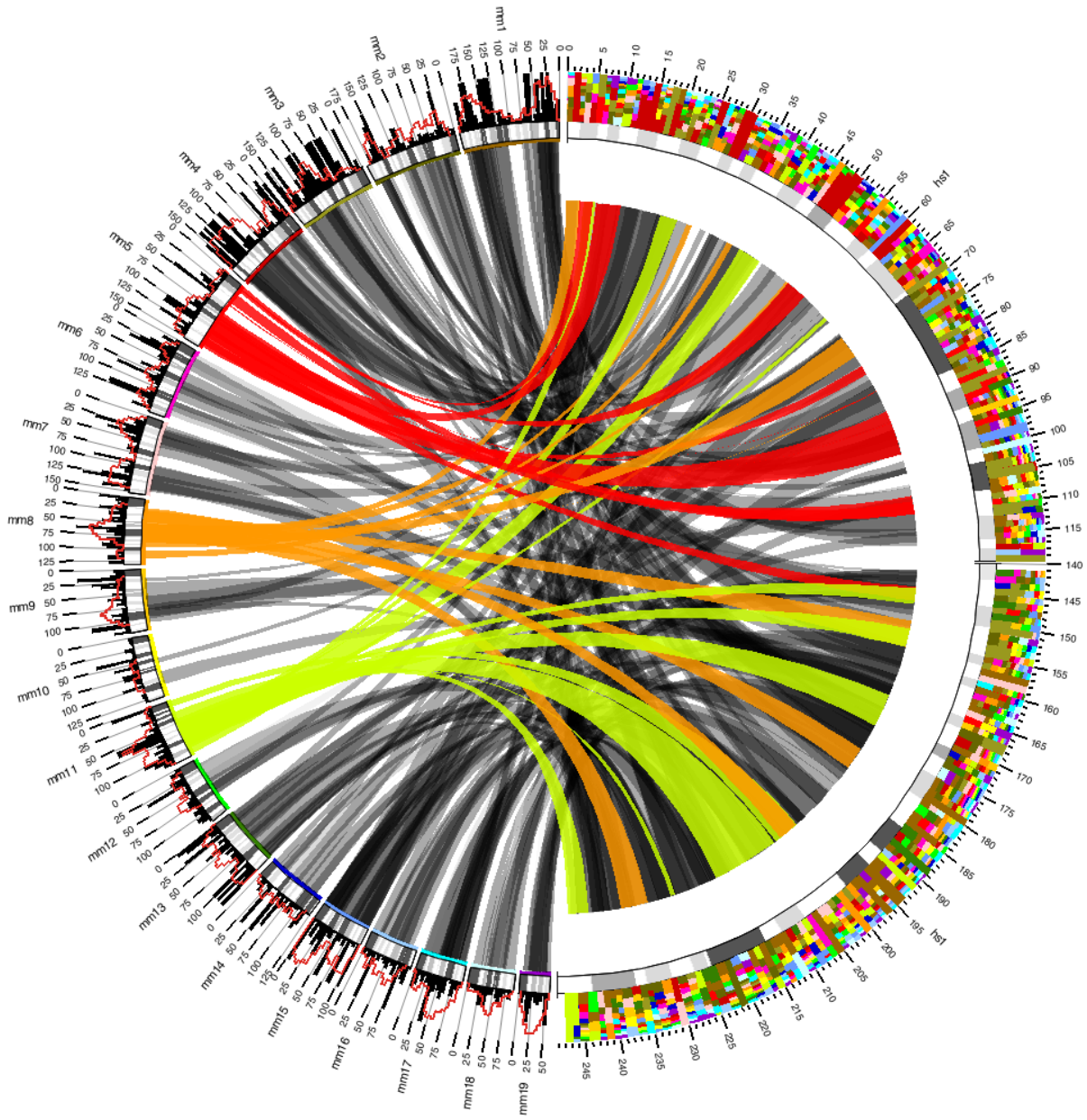

FIGURE 15

The outer stacked histogram is normalized – the total of each bin is 1.

Together with the inner density histogram placed on hs1, we see the absolute size of links between a window on hs1 and the mouse chromosome with greatest similarity, as well as the relative fraction of links between the window and each mouse chromosome.

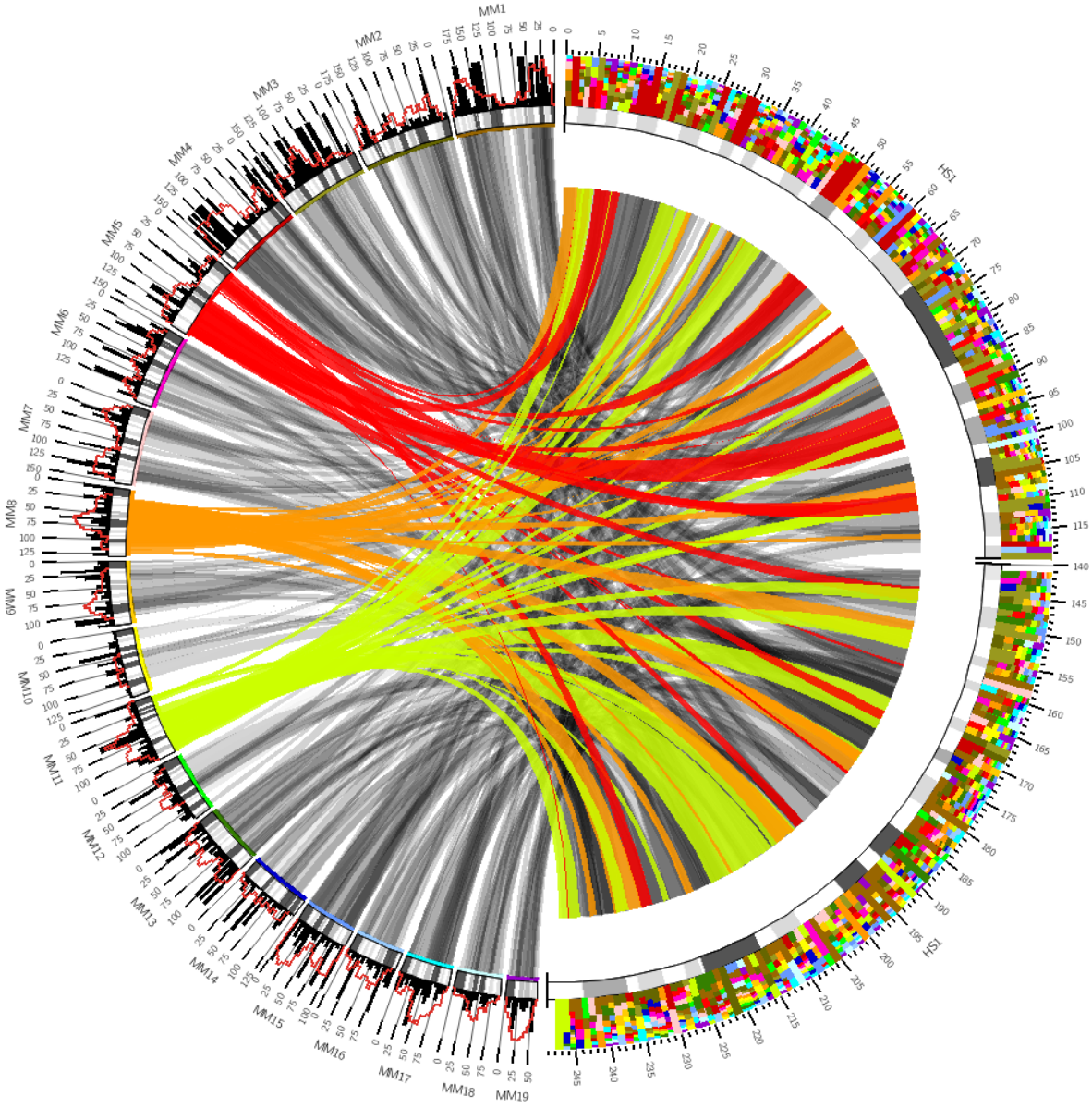

FIGURE 16

The outer stacked histogram is normalized—the total of each bin is 1.

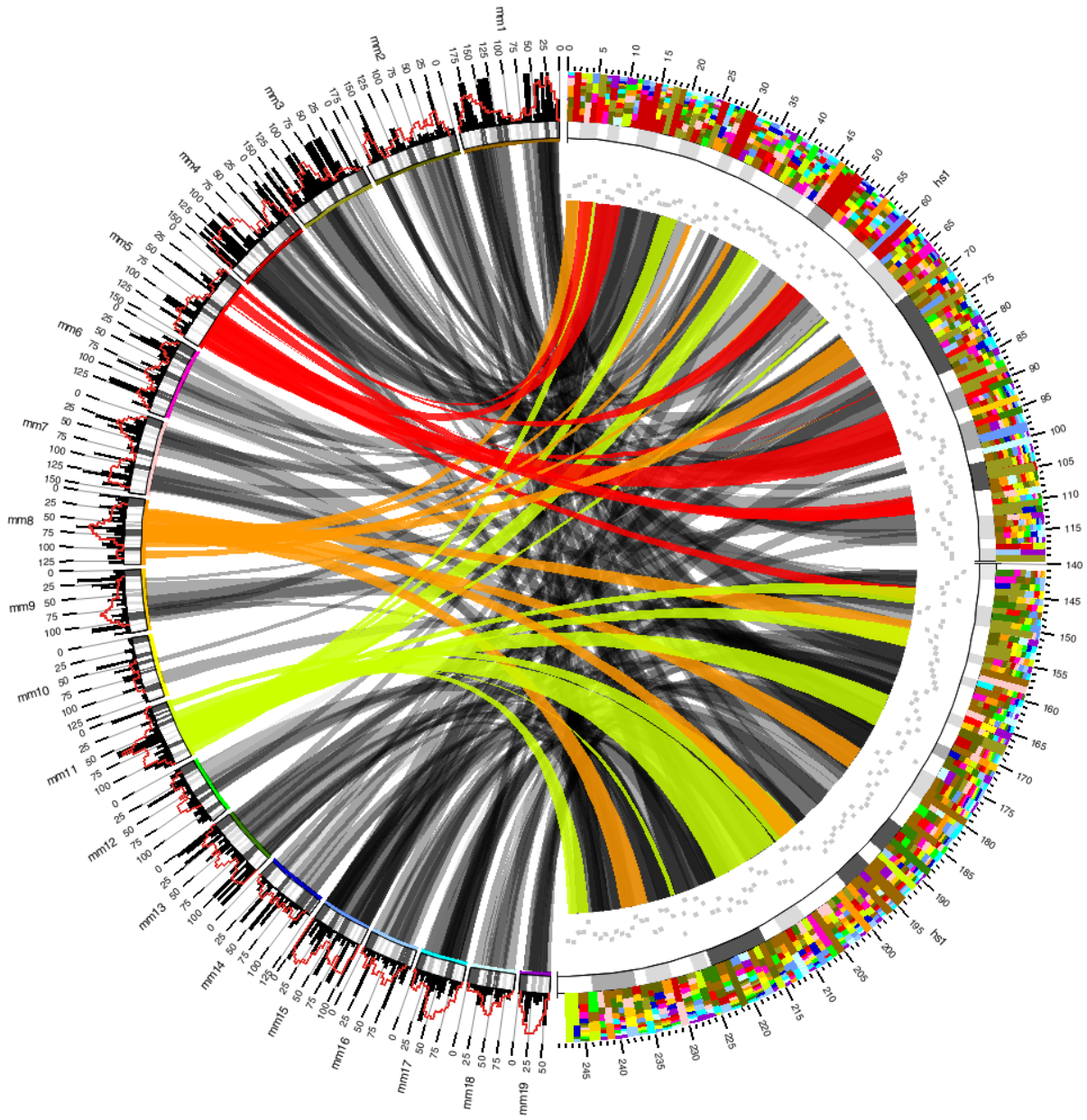

FIGURE 17

A scatter plot is added to the figure to show average conservation within 1Mb bins on hs1.

Rules are applied to the plot to color glyphs based on value.

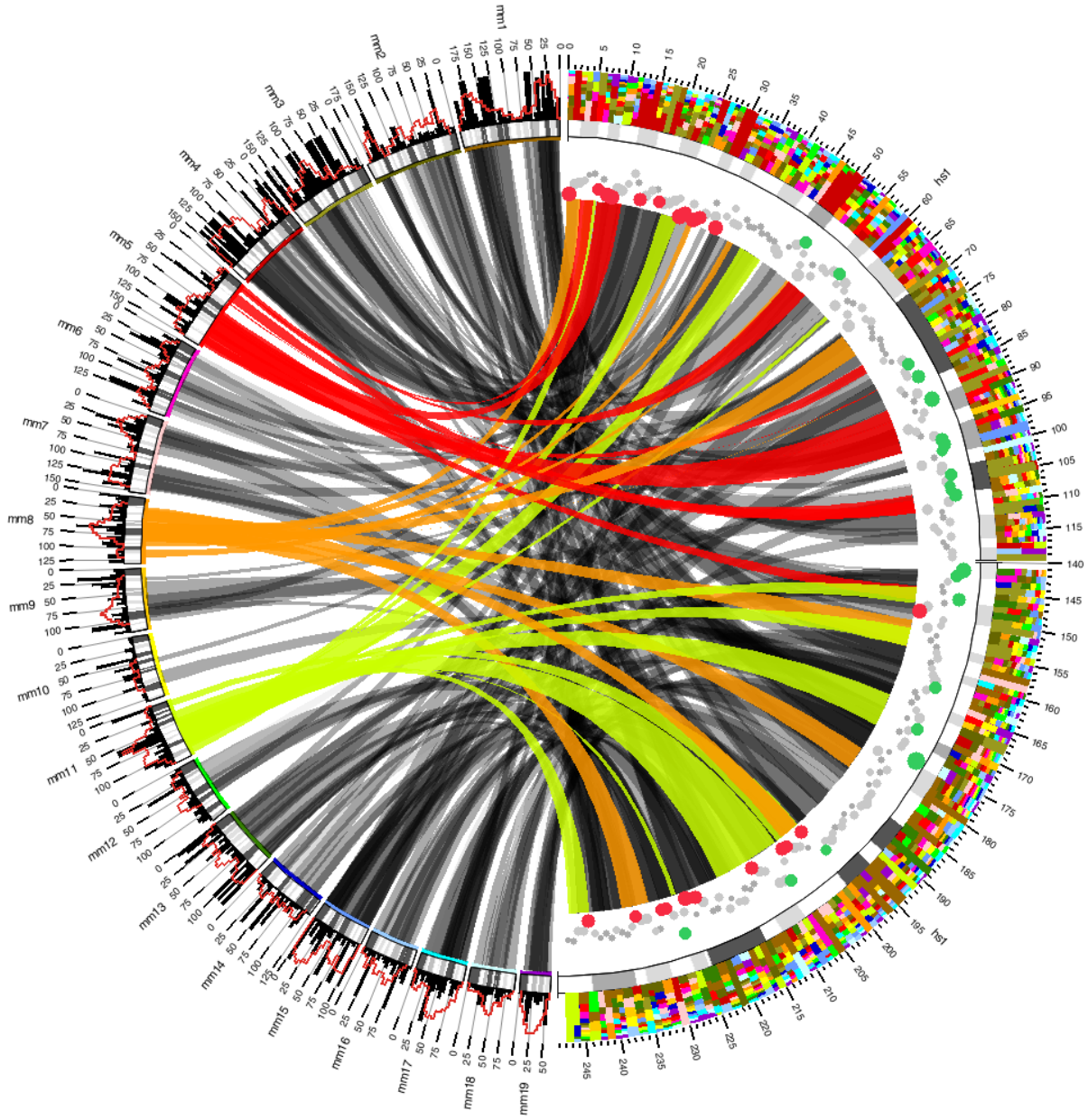

FIGURE 18

Glyph size is made proportional to the deviation of the data point (distance to average).

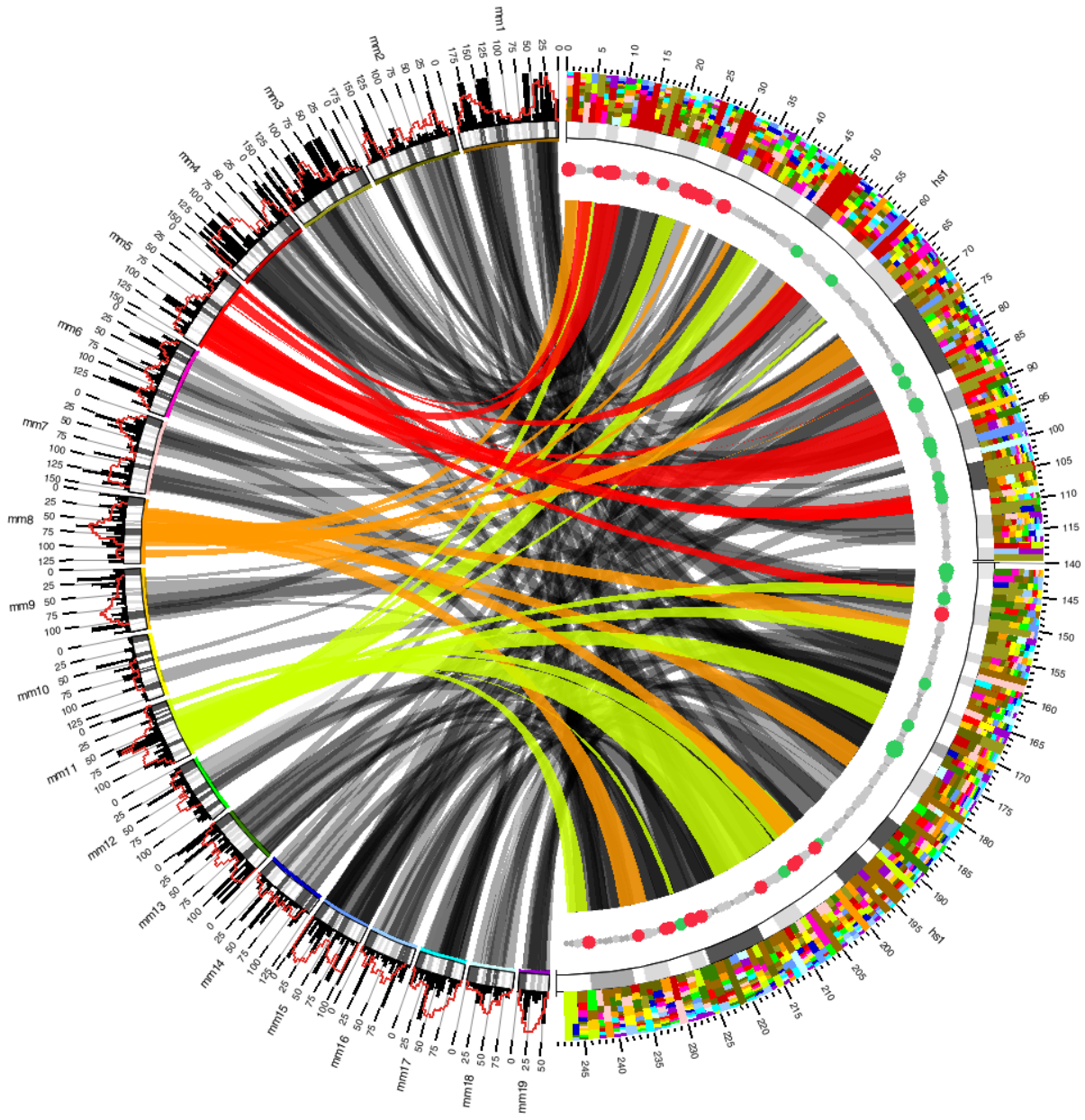

FIGURE 19

By mapping value onto glyph size and then placing all the glyphs at the same radial position (by changing data values), a glyph track is created. Stacking such glyph tracks can create very interesting (and attractive) visualizations.

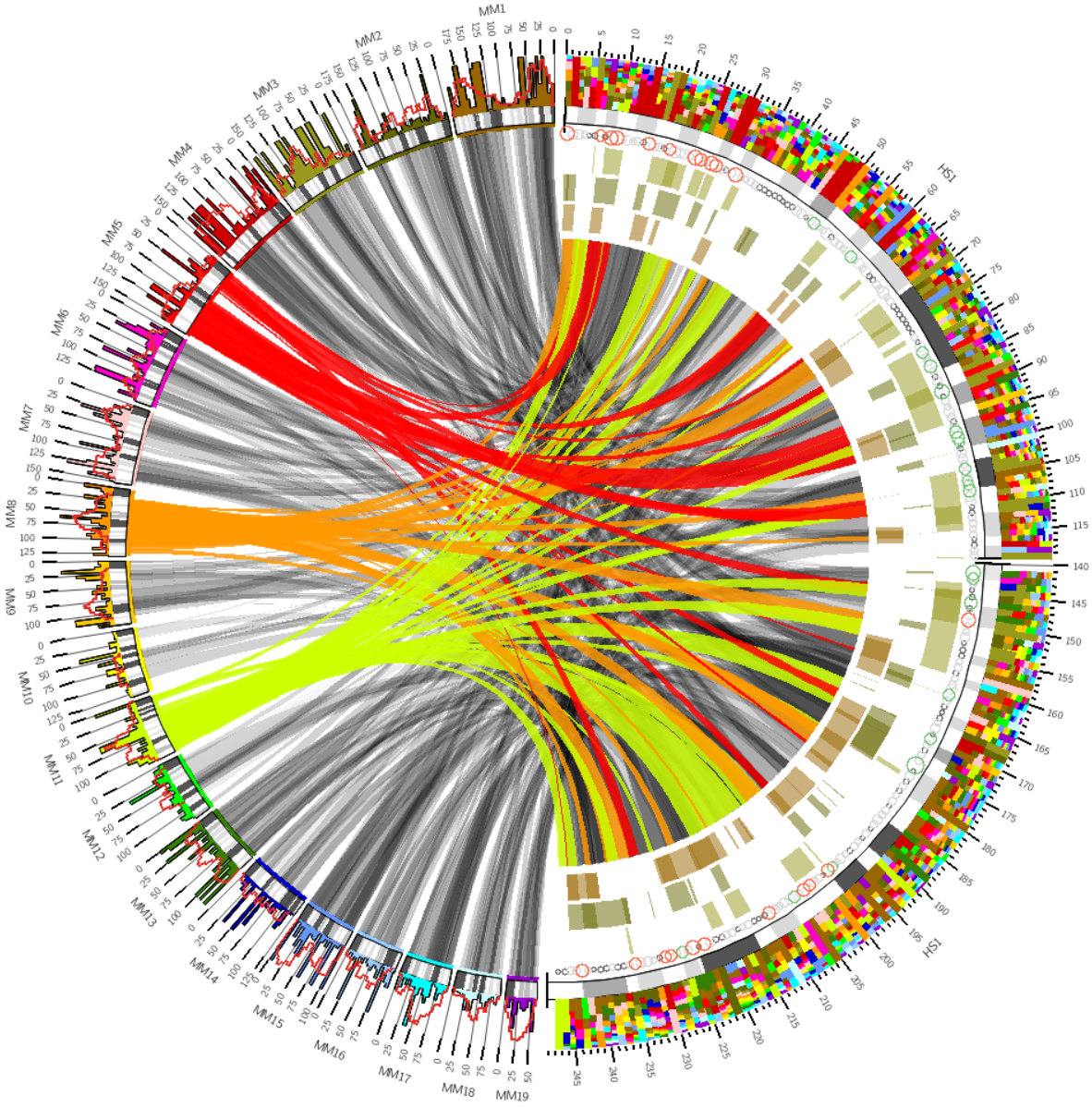

FIGURE 20

3 automated heatmaps.

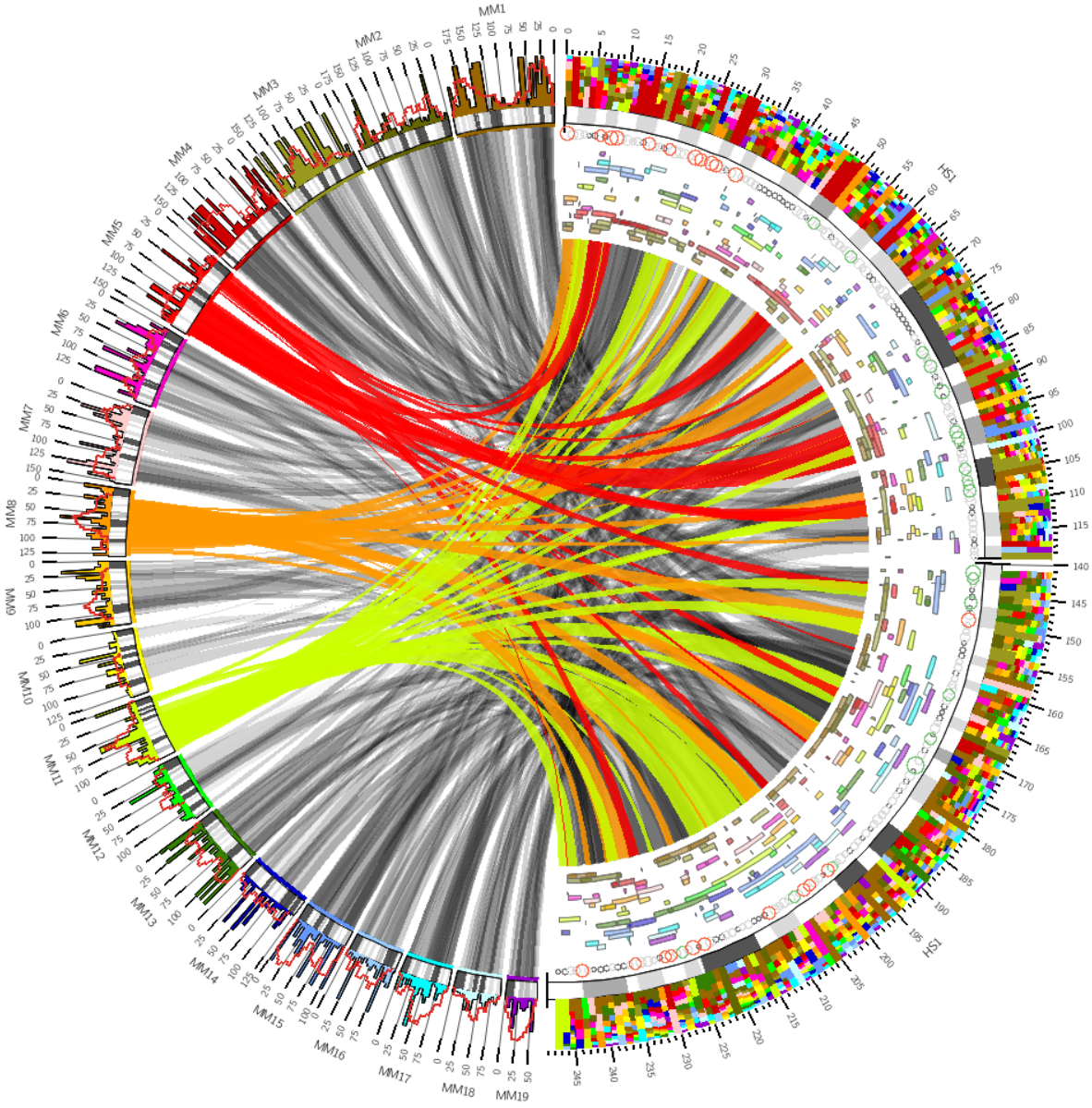

FIGURE 21

19 automated heatmaps.

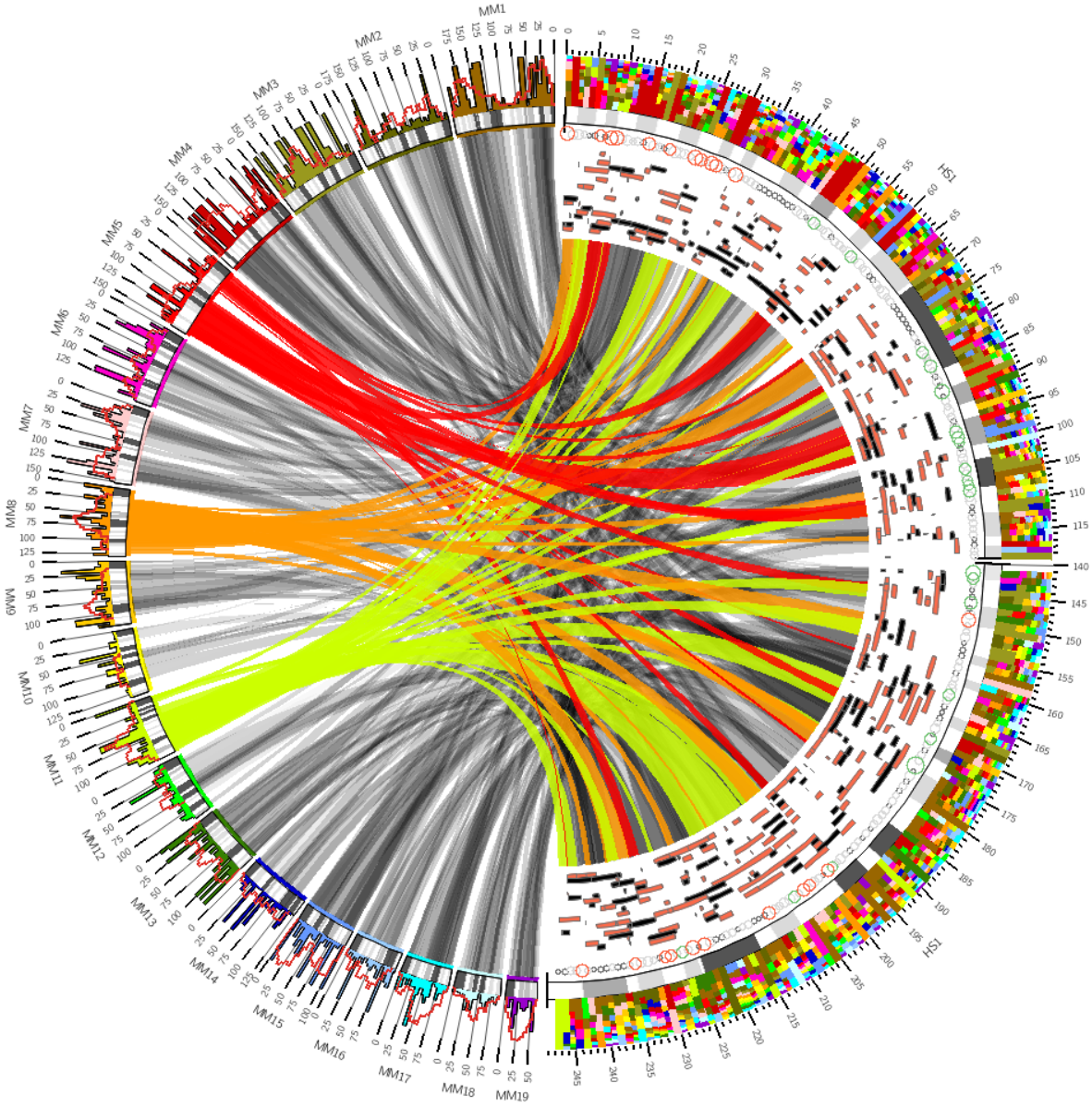

FIGURE 22

19 automated heatmaps. Tiles are colored alternating black/red based on heatmap counter.