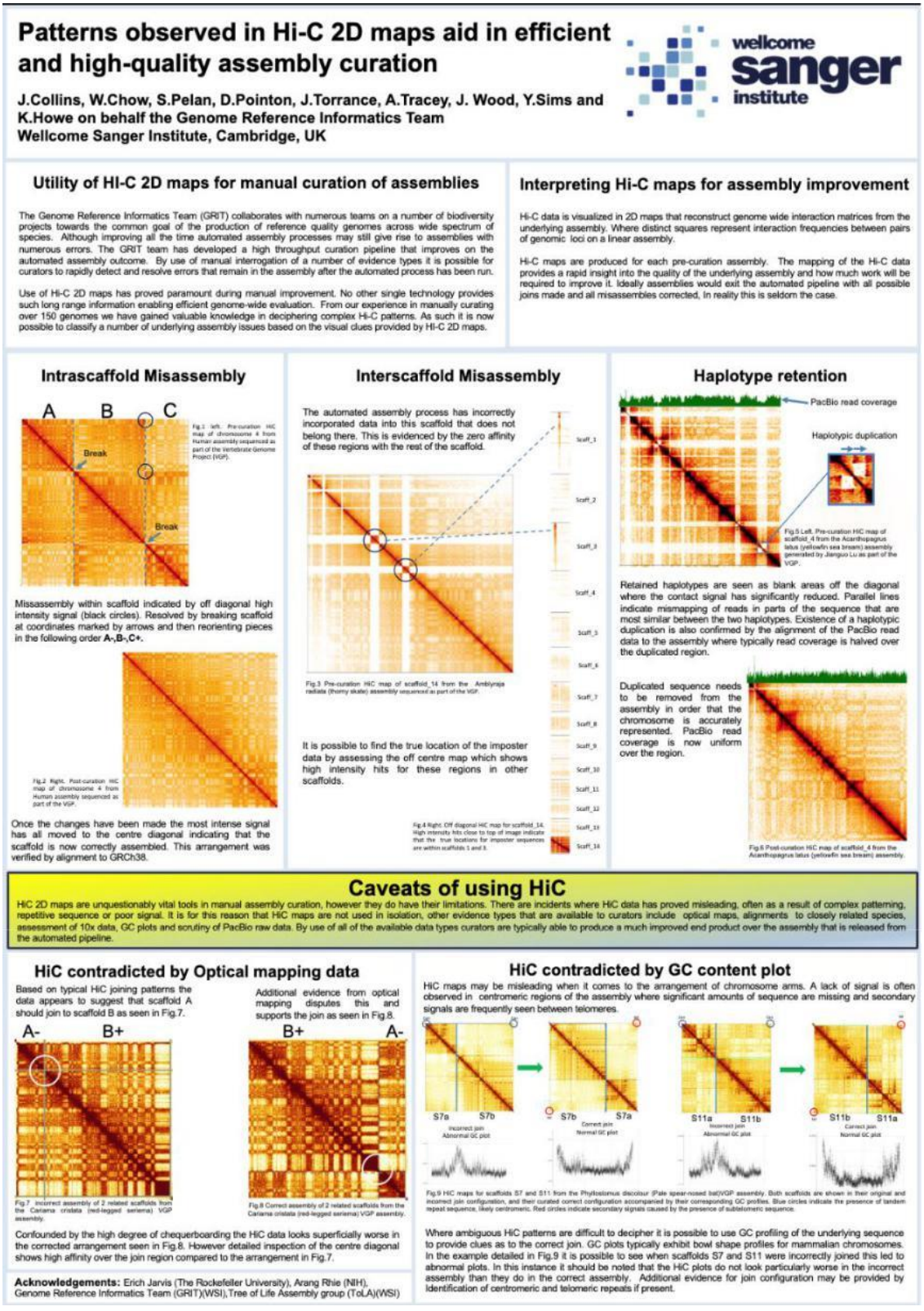
Internship journal 2

Project Background

Chromosomes of almost all species to date have higher-order organizational features such as interaction domains or loops. These structures are often associated with biological function and form unique visible patterns on genome-wide contact maps generated by chromosome conformation capture methods such as Hi-C. While HI-C is the focus of this project, HI-C can determine the average frequency of contacts between chromosomes and DNA segments within the genome, calculated for hundreds of thousands of cells, and in doing so, has the significance of explaining the 3D structure of various chromatin in a variety of organisms. High-quality and complete reference genome collections are the basis for applying genomics to biology, disease, and biodiversity conservation. Genomics is the basis for using genomics to biology, disease, and biodiversity conservation. Genome assembly is putting nucleotide sequences into the correct order and orientation.

Accurate assembly of animal and plant genomes at the chromosomal level, these large and complex animal and plant genomes require long-distance linkage data (i.e., Hi-C data) to bridge the assembly gap accurately; between sequences at the allelic genome level(Dekker, 2008). The automated assembly scaffolding process may incorrectly incorporate data that does not belong in that scaffold. Data that do not belong there. In these cases, clear boundaries can be seen on the Hi-C heat map. The image below shows the problems that can be encountered in the HI-C heatmap. You can see that there are many types of issues, including assembly errors, monotype retention, etc., but the fixes are very similar in terms of computer vision; find the wrong point and determine the location in the image, backtrack back to the file that generated the image, and use the image location information to locate the text in the, aside from this method, to achieve a high degree of chromosome accuracy, PFR has also developed a shinyR program called HiCSuite, which joins sequences semi-manually by visualizing contact maps.(this figure may cannot look clearly, hence the origin image attached in email)。



(Wu, 2022)

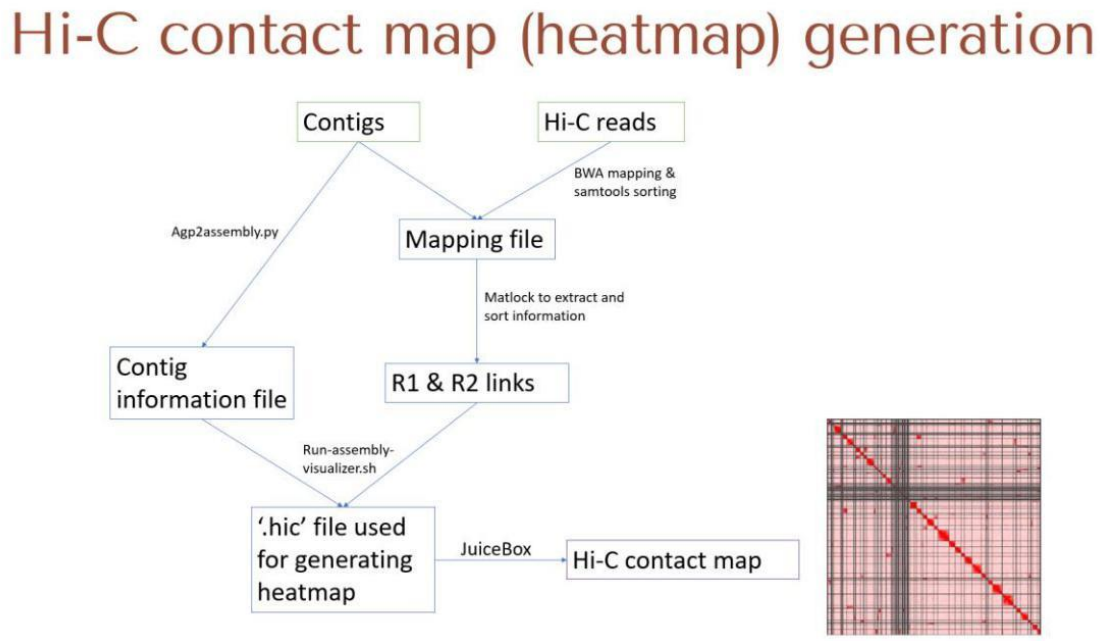
While this method allows for accurate concatenation of sequences, it is time-consuming when scaffolding large numbers of sequences and can introduce human error. In addition to scaffolding, assemblers can also generate assembly chimeras at the Contig level, leading to incorrectly linking

sequences from distant regions or even different chromosomes at the Contig and chromosome levels. Automated image processing was therefore included as a possible approach.

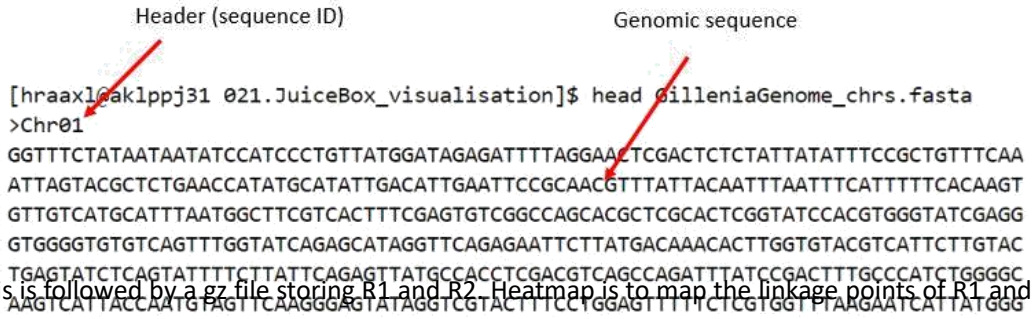
At the beginning of the project, we considered edge (boundary) detection as a catalog of computer vision, including various mathematical methods designed to identify edges and curves in digital images where the image's brightness changes sharply (Matthey-Doret et al., 2020). Drastic changes. Some of these methods have been tested in the automatic processing of Hi-C heatmaps to detect boundary structures to discover functional elements in 3D contact. (Yoon, Chandra & Vahedi, 2022)However. These have not been applied to genomic scaffolding applications. In addition to boundary detection, search strategies such as greedy algorithms are needed to efficiently search for the best When there are many Contigs, search strategies such as greedy algorithms are also required to search for the best candidate sequences efficiently (He, Li, Ye & Zhong, 2012). In this project, the idea is to use an image-processing-based approach to curate allele-level chimeras automatically and scaffolds using Hi-C linkage information and Automatically curate allelic genomic and scaffold sequences using Hi-C linkage information.

Project Details and Progress

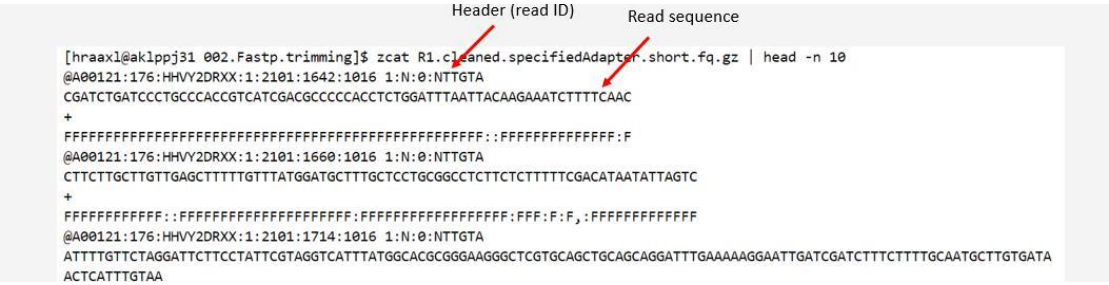
Firstly, it's about how the HIC heatmap is generated, by which files, and what steps they go through.



There is the .fasta file, which stores the sequence of genes, the contigs, generated from the raw whole genome sequencing reads; the contigs that need to be post folded at the chromosome level using Hi-C data, can be hundreds or thousands of contigs.

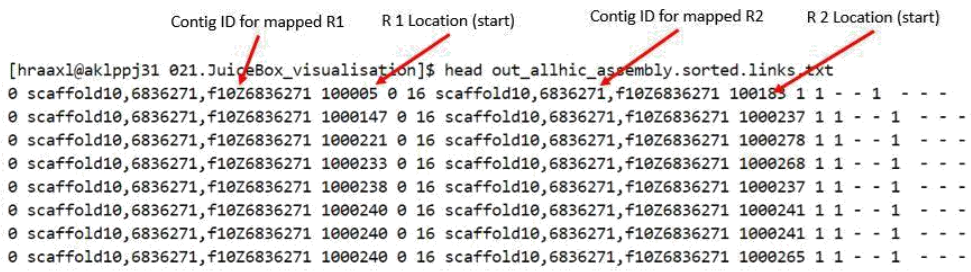


This is followed by a gz file storing R1 and R2. Heatmap is to map the linkage points of R1 and R2. The data is used to construct a scaffold of genomic sequences, which can be millions/billions of sequences.

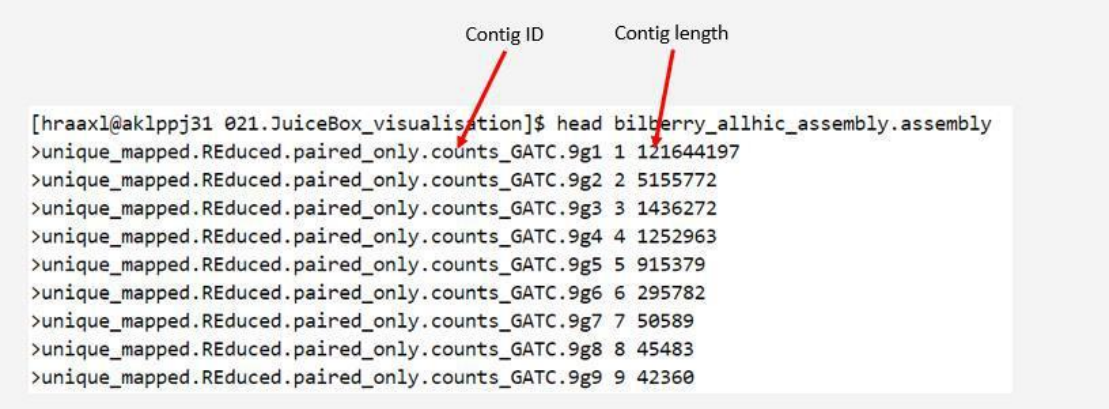


And after the sequence and R1 and R2 are obtained, the Bam file can be obtained by mapping and is the output file that matches the Hi-C reads to the genomic sequence; storing the position, length, and ID of R1 and R2 in the sequence; each read is found to match the best matching region on the genomic

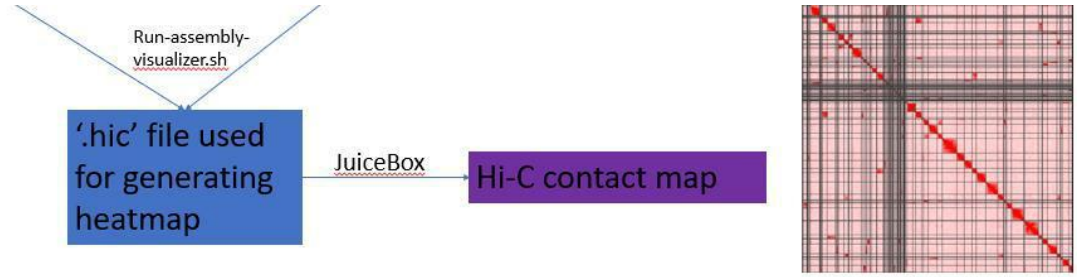
The bam file is simplified and converted to give a sort.out.txt file storing the R1 and R2 link information. Each matched R1 and R2 information is stored in a single line, efficiently representing one genomic interaction/link and containing the position and ID.



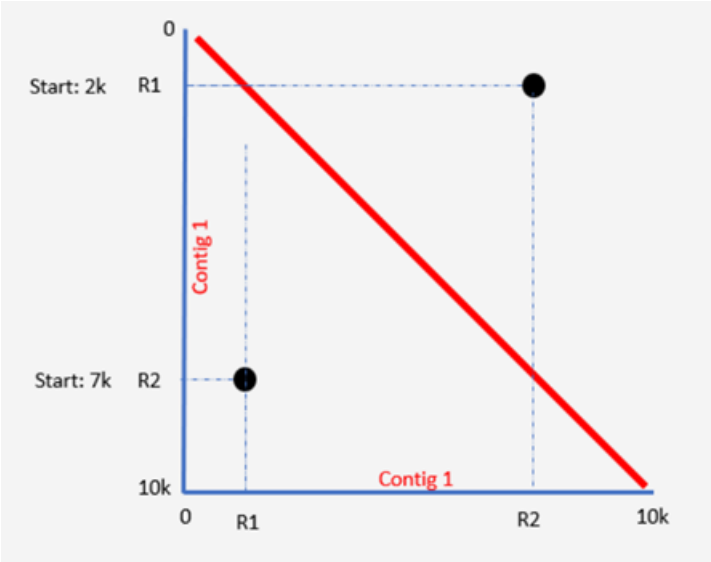
This is followed by the assembly file, which generates the information for the contig file, and also contains the length of the contig;.

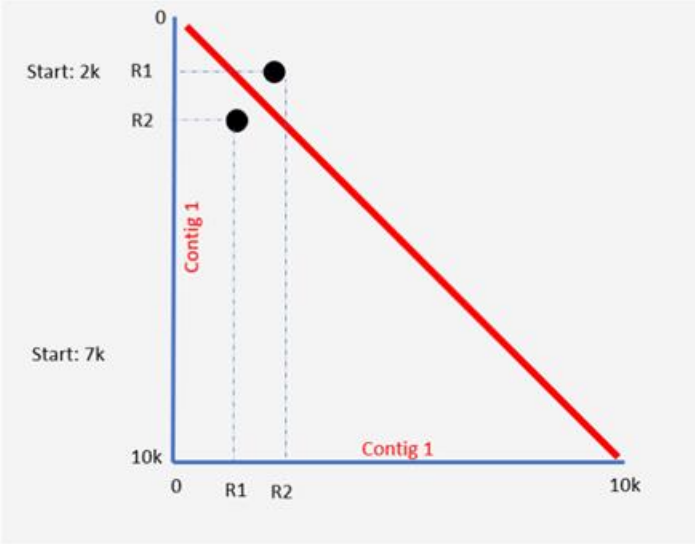


Finally, the HIC file is generated from the out.txt file containing the link information and the R1 and R2 locations, and the assembly file containing the contig information, the exact format of which is not known at this time. The final hic file is converted into a heatmap by a HIC drawing tool called jukebox, but we don't know .hic file type and format yet.



Then will be the reason that there's a clear diagonal in the heatmap. The R1 & R2 link file (.hic) only has the start mapped locations for R1 & R2; in this figure, R1 is around 2k position and R2 around 7k position; hence there are two dots there, R1 and R2, which are close to each other will have a higher frequency, and far away for each other will have a lower frequency. The closer the mapped R1 & R2 are, the closer the dot is to the diagonal. A lot more dots are sitting close to the diagonal, indicating a much higher frequency of interactions between the genomic regions that are close to each other, hence will generate a diagonal.



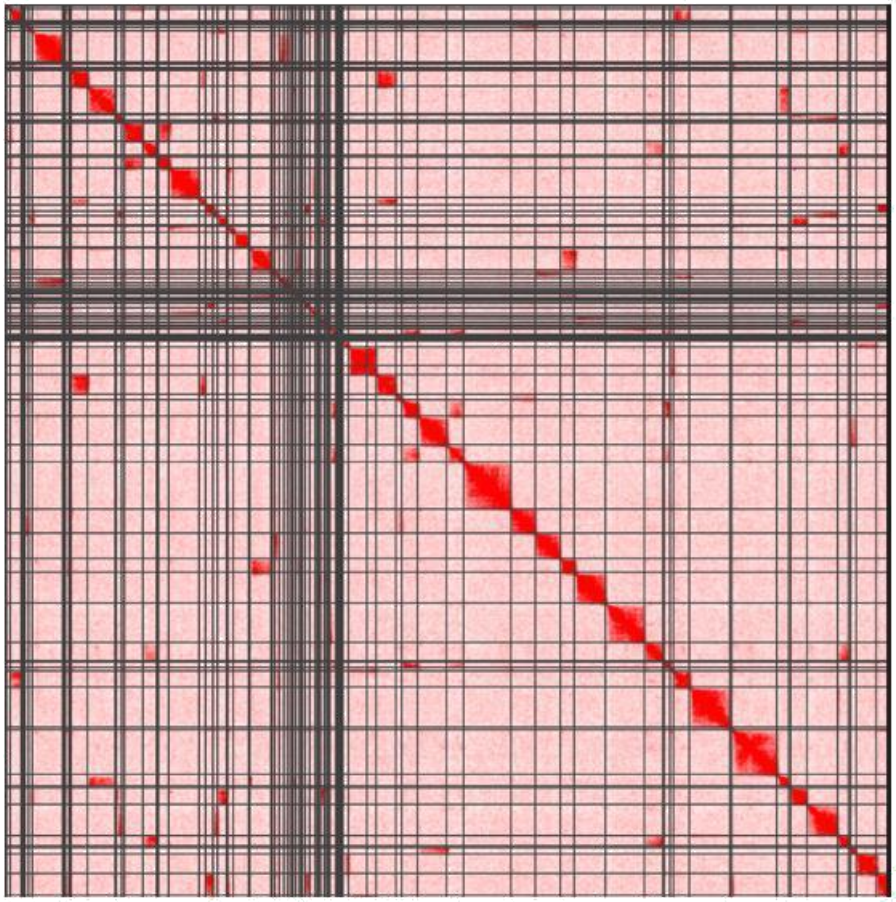


This is an example of a fish heatmap(figure next page), a combination of raw sequencing reads with alleles.

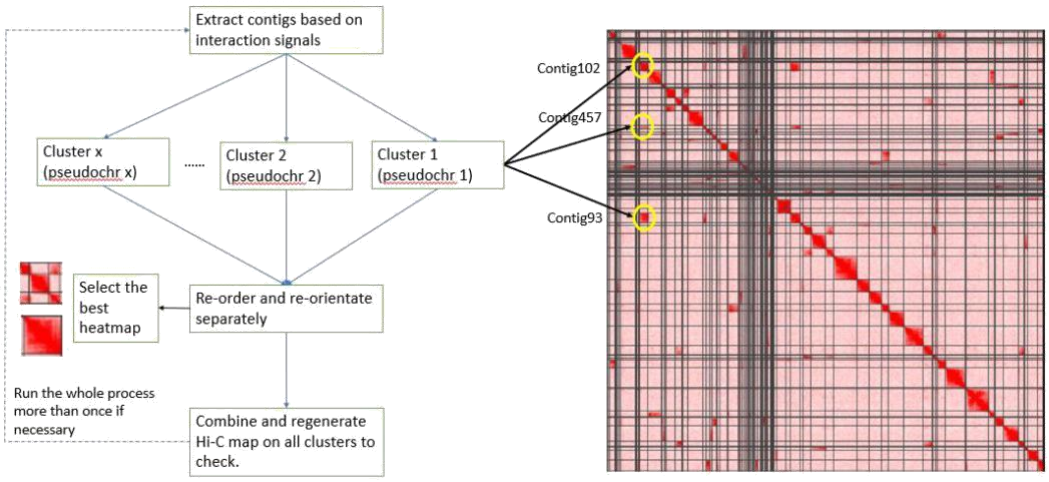
The order and orientation of the alleles are random on the Y-axis and the same on the X-axis (left to right).

The grey line separates the two alleles on the map.

Strong signals (red squares) can be seen due to the frequent contact between the two allelic groups (circled).



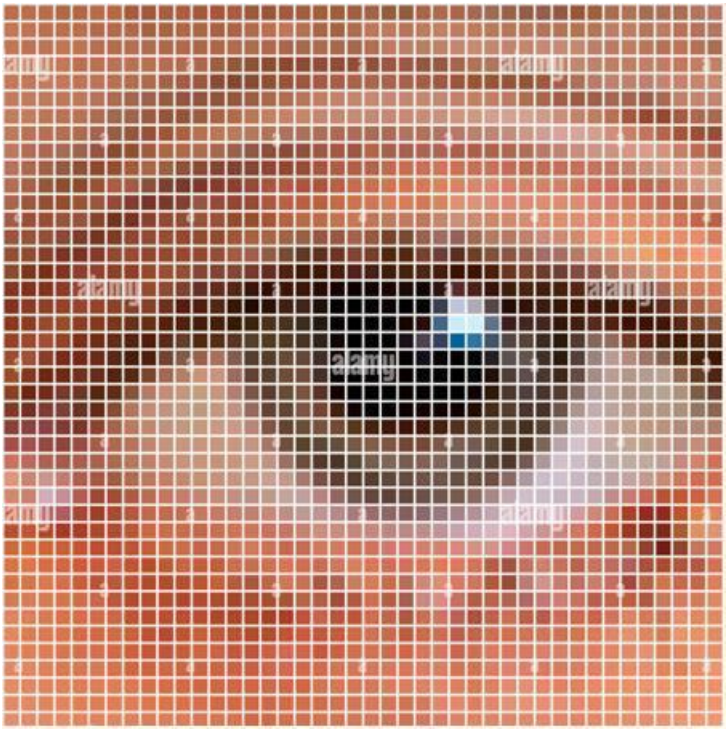
It can be noted that there are also many unorder regions on this image. If there are significant red squares at the same line as the squares on the diagonal, identify them as unorder. We use the image process to identify these squares and location them; this is the means of extracting contigs based on interaction signals; we will fish the courts those in the same lines out, backtrack to the .hic file or bam file, and according to that will extract lots of clusters, cluster 1 includes three red squares; do a reorder on each of them, etc. Since we don't know the order of the three squares, we may do more than one reorder and try various combinations until we get the correct order; every order should use a heatmap to test(i.e., try 123,321,231 …).



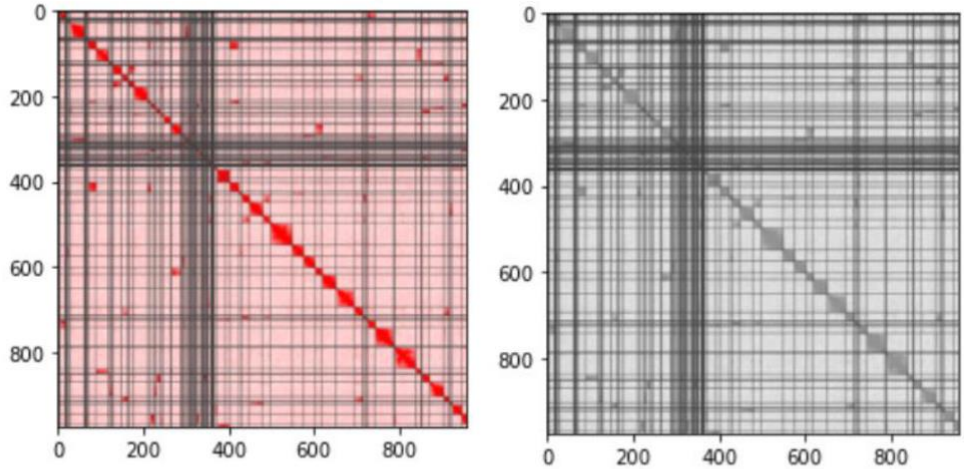
We need to do this for all the clusters until we have repaired them and finally joined them.

Or we could identify all the errors and put them in the right place, but we think it would be harder to find the right place.

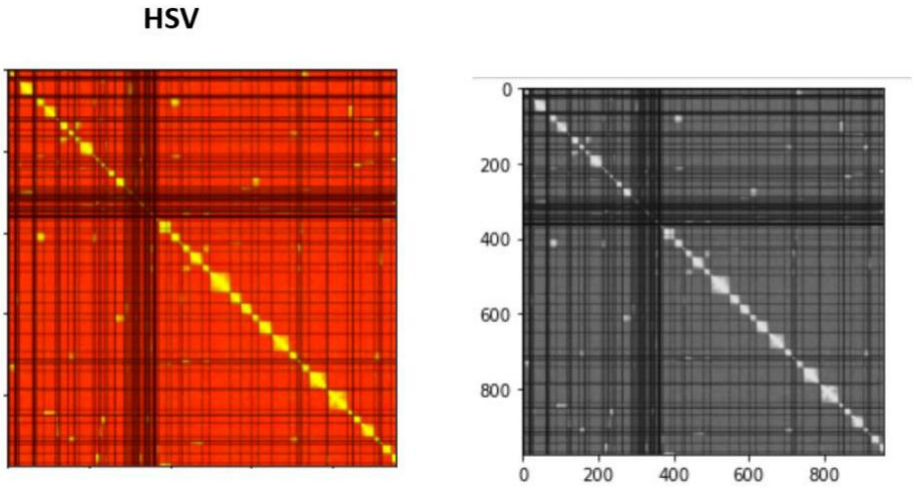
After that, a little briefly introduce image processing. All images are made up of individual image pixel points; each small square in this image represents a pixel point; an 800*800 image is more precise than a 400*400 image because there are twice as many pixels, and These pixel points are stored in a matrix, with pixel points having values between 0 and 255. A color image usually has three or four channels or, in the case of three channels, three matrices for RGB, i.e., red, green, and blue.



Looking directly at the columns, this HIC has 975 rows and 957 columns, which means that each matrix stores 933075-pixel points; this is a color picture. Therefore, there are three such matrices. Some image processing algorithms only accept single-channel images for processing because color images contain so much information. A single-channel matrix is much easier to process than a three-channel one, and many packaged OpenCV functions only accept greyscale images as input. As this image shows, there is only a 975*957 matrix after it has been turned into a greyscale image. The matrix values are still between 0 and 255, with 0 representing black, 255 representing white, and varying degrees of grey between 0 and 255. Because the diagonal lines of this grey map, the background, and the division lines are too similar in color, there may be errors in processing.

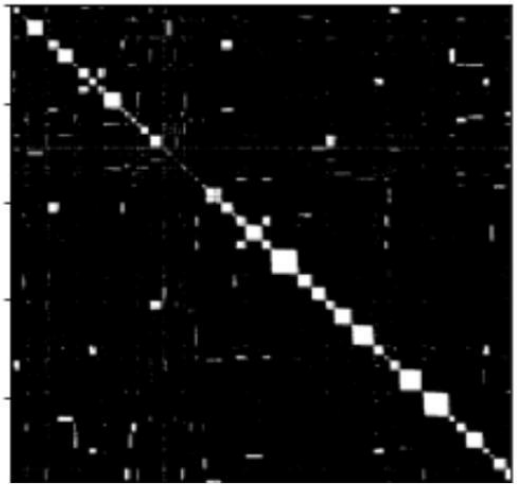


I therefore did an HSV conversion of the original image, which is simply a way of highlighting the colors, and then a much clearer greyscale conversion afterwards.



Sometimes even after greyscale, the image still has too much information stored in it, so it needs to be binarized, i.e., all the colors in the image are set to 0 or 255; the method is simple, set a threshold between 0 and 255, anything more significant than that value is set to white, anything less is set to black, in this case, I put it to 150 so that all the small red squares in the original image are In this case I set it to 150 so that all the small red squares in the original image are extracted,and

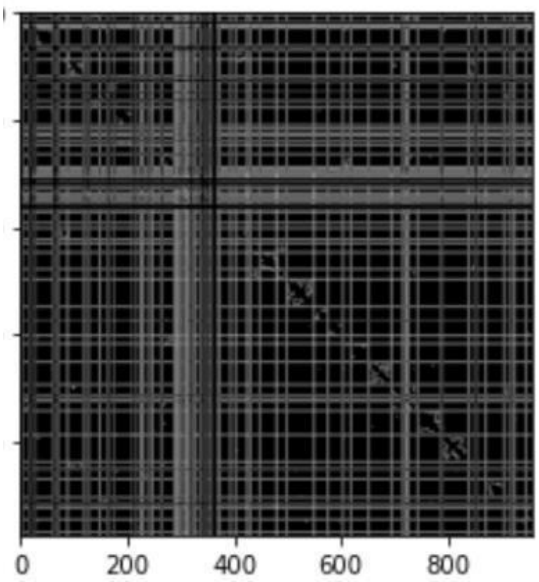
the rest of the image is black.

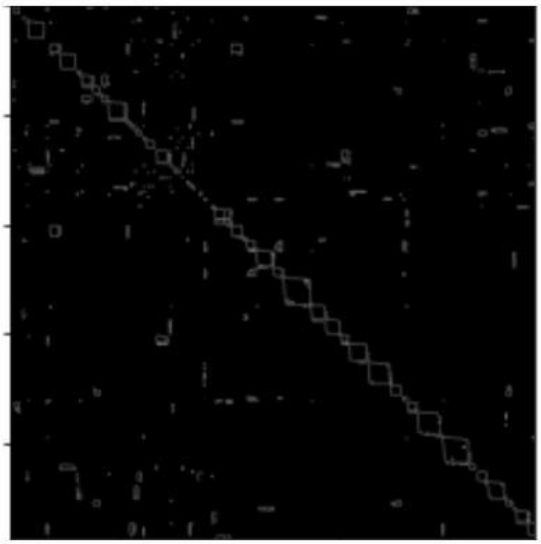


```
i = threshold(imgnn,150)
```

An edge is a set of connected pixels that lie on the boundary between two regions, and edge detection is the most common method for detecting significant discontinuities in grayscale (H.JaseemaYasmin & Mohamed Sadiq, 2012), this is the reason that we consider border detection firstly. This is a Canny border detection image of the original image and the binarized image, but I am still not sure how this detection will help this project to find the error points, including edge detection; there is inevitably image smoothing, which uses some simple convolution operations. For example, Gaussian smoothing is used in canny edge detection, where a convolution kernel is set to 3*3(in terms of Gaussian filtering, the value of the seed is set to be larger the closer it is to the center).

And a 3*3 region is drawn in the image matrix, and then the 3*3 kernel is used to weight the average of the 3*3 region in the image, and the resulting value replaces the original value in the image. Sliding processes the whole image, smoothing those tiny points before identifying the boundaries, which may impact images such as the HIC heatmap. These small squares in the heatmap are also critical and require much accuracy.





INPUT IMAGE

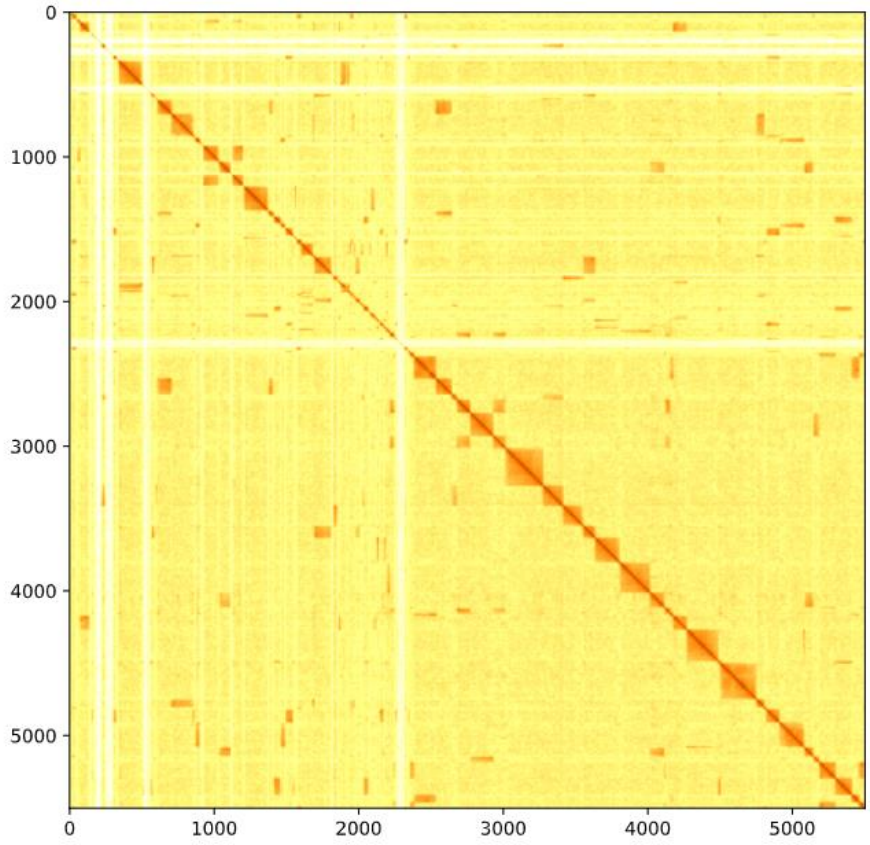
18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

0.6	0.8	0.6
0.8	1	0.8
0.6	0.8	0.6

Other edge detection methods also have morphology methods like Opening and Closing operations processing that will erode or inflate these small regions and affect pixel points. Hence, we had a meeting to talk about questions.

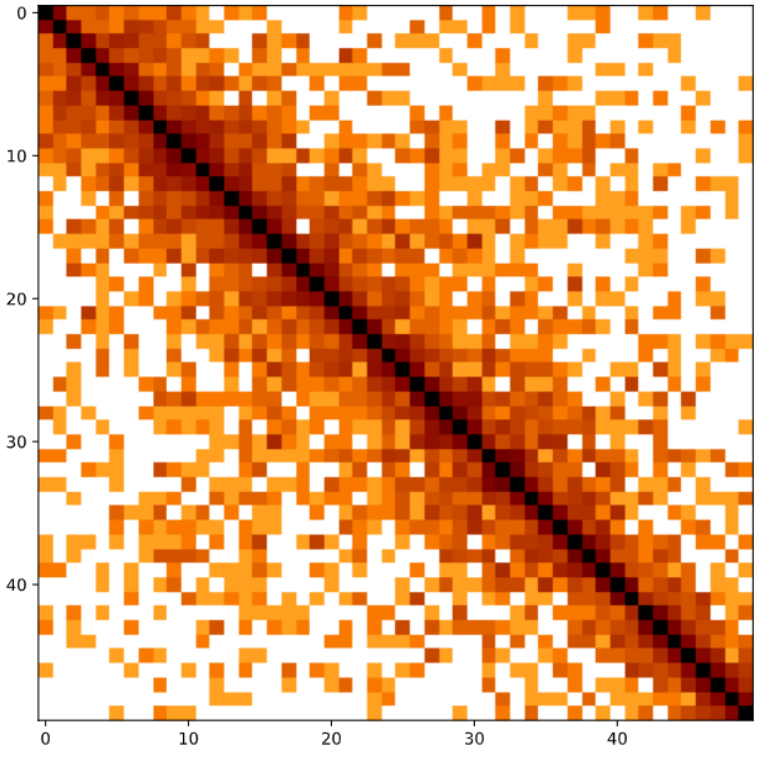
In addition to the above, we have also encountered issues such as the size of the images varying and, if resizing would also damage the images, how to backtrack from the image location in case the interaction point is larger than the pixel point of the image which would be difficult to handle. So, the plan for the next few weeks is to look at some of the online tools for generating heatmaps, study their source code and try to get to grips with the code for generating heatmaps. If we can develop the images ourselves, then backtrack, and image size will no longer be an issue; we are also trying to figure out the format in the hic file, which will probably be the best input for generating the heatmap file.

After discussion, we decided to generate our heatmap using the code instead of generating it through online software; the advantage of doing so is that we can scale at different resolutions and scales according to the original data. For example, the second and third pictures below are the zoom-in of a specific contig and the zoom-in of a specific picture area, respectively, and then regenerate the heatmap.



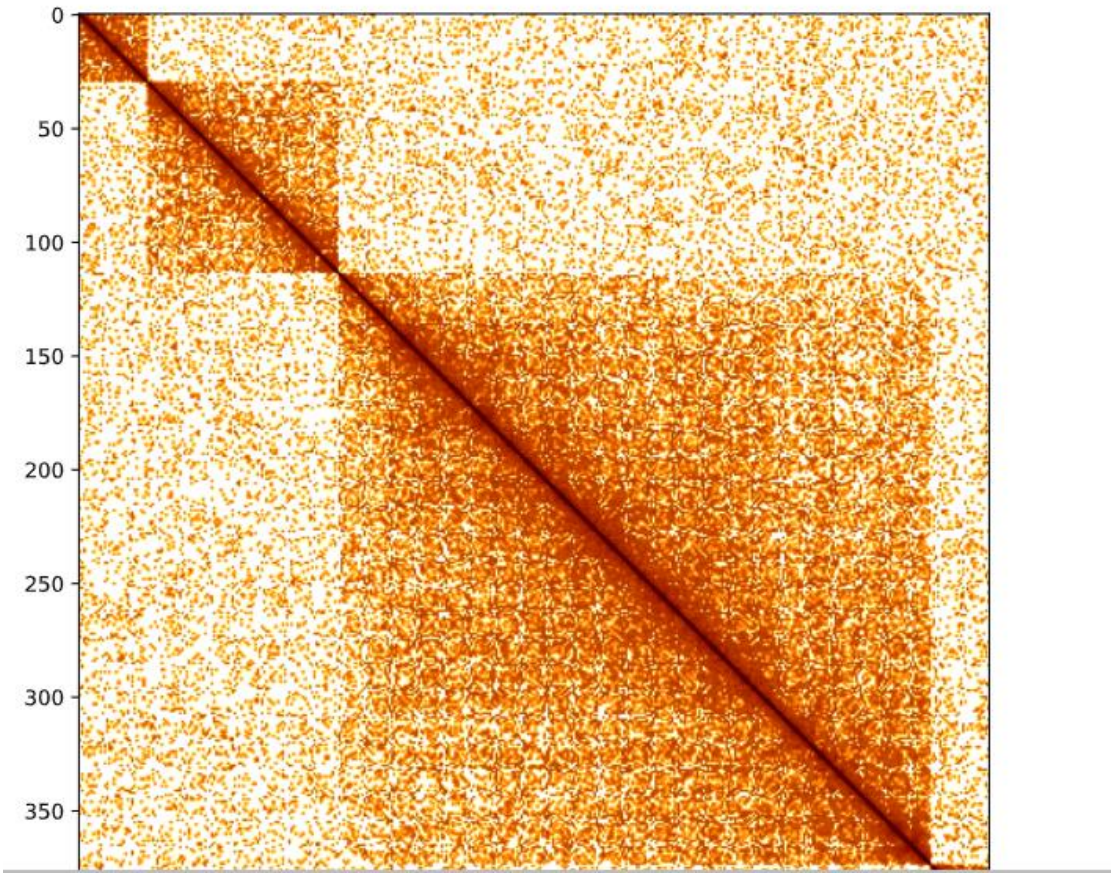

```
region = 'scaffold_251_pilon_pilon:15000000-20000000'
mat = c.matrix(sparse=False, balance=False).fetch(region)
plt.figure(figsize=(8, 8))
plt.imshow(np.log10(mat), cmap='afmhot_r',norm=colors.PowerNorm(gamma=0.35))
```

```
<matplotlib.image.AxesImage at 0x7efc78410290>
```



```
## zoom in specific range in heatmap
mat = c.matrix(sparse=False, balance=False)[2900:3300,2900:3300]
plt.figure(figsize=(8, 8))
plt.imshow(mat ** 0.2, cmap='afmhot_r',norm=colors.PowerNorm(gamma=0.35))
```

```
<matplotlib.image.AxesImage at 0x7efc691117d0>
```



After that, since we need to find the darker red plot in the same column or row, we thought it would be helpful to split the whole heatmap according to the contig id. Using chromosight's package, I pulled out the relevant information (data frame) containing the contig id and each contig's pixel start and end points.

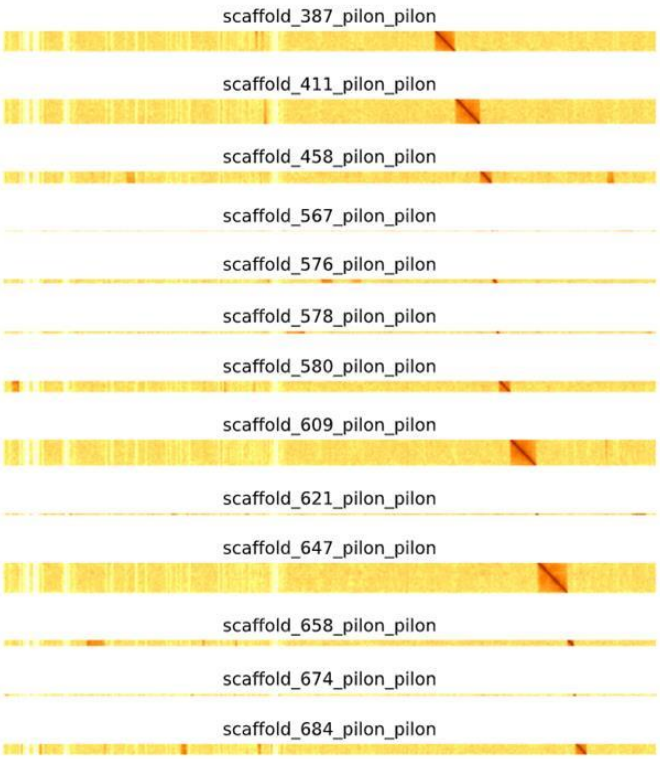
[102]:

	name	length	start_bin	end_bin
0	contig_1_pilon_pilon	149114	0	2
1	contig_10_pilon_pilon	2606	2	3
2	contig_1000_pilon_pilon	100064	3	5
3	contig_1004_pilon_pilon	1806	5	6
4	contig_1005_pilon_pilon	1677	6	7
...	...	...	...	...
561	scaffold_779_pilon_pilon	11052189	5186	5297
562	scaffold_780_pilon_pilon	11210716	5297	5410
563	scaffold_796_pilon_pilon	5020744	5410	5461
564	scaffold_818_pilon_pilon	3485666	5461	5496
565	scaffold_976_pilon_pilon	646342	5496	5503

566 rows × 4 columns

Then I transformed them into a list and a dictionary and transformed the original map into 566 horizontal partitions according to each contig's bins start and endpoints(as show above).

```
{'contig_1_pilon_pilon': array([[1320,   23,    0, ...,    0,    0,    0],
        [  23,  553,    0, ...,    1,    0,    0]], dtype=int32),
 'contig_10_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1000_pilon_pilon': array([[2, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1004_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1005_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1006_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1015_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1016_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1018_pilon_pilon': array([[0, 0, 0, ..., 2, 1, 0],
        [0, 0, 0, ..., 2, 0, 0],
        [3, 0, 0, ..., 3, 0, 2]], dtype=int32),
 'contig_1020_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 2, 0],
        [0, 0, 0, ..., 1, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 1, 0],
        [1, 0, 0, ..., 0, 0, 0],
        [0, 1, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1021_pilon_pilon': array([[0, 0, 0, ..., 0, 1, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 1, 0, 0],
        [0, 0, 0, ..., 1, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=int32),
 'contig_1022_pilon_pilon': array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 1, 0],
        ...,
```



Thanks to the dictionary, I can find the 2D Array of a specific contig based on the contig id (Key), and I can generate the image of a specific contig based on the 2D Array; similarly, I can know the specific ID of the contig based on the 2D Array of the generated image (Value).

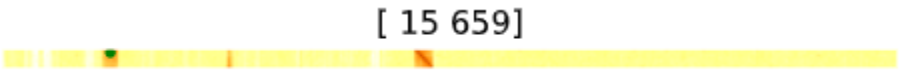
```
scaffold_1465 = fish_dic.get('scaffold_1465_pilon_pilon')
generatemap(scaffold_1465)
```

```
print(get_contig_id(splited_matrix[30],fish_dic))
```

```
contig_1015_pilon_pilon
```

Since we need to know which two contigs are causing the unorder problem, I wrote a function to determine which contigs the pixel index belongs to based on the input x-axis pixel index. After that, we need to locate the pixel point where the darker red plot is, so we can use the above function to find which contig it belongs to based on the x-axis pixel index, the image of the row itself can use the dictionary to find its contig id, and the locating work is done. As shown in the figure below, the contig image itself is contig scaffold 1465(use dictionary find), and the row pixel index 659 is obviously a prominent red region, use the function to find the contig id represented by this index is contig 187. Therefore, we can determine that there is a problem with the connection between these two items.

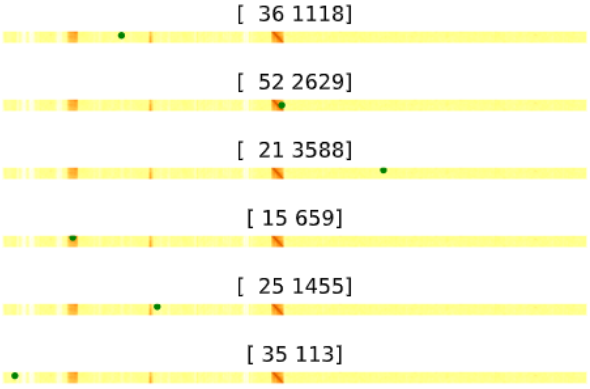
(the x-axis index 659(green point) belong to contig 187).



```
find_location(fish_matrix,fish_chro_info,fish_dic,659) # if we know the red square x a
```

```
contig_187_pilon_pilon
```

Now to automate the search for these prominent red regions, I decided to use 2D Array's peak detection to do a search. One of the common challenges in signal processing is to detect significant points in time or space, such as peaks, which are locations on an image where all surrounding pixels have smaller values. These points represent local maxima or minima that usually have a proper meaning in a fixed example; detection is already not trivial in the 1D case. However, it becomes more complicated if we want to do it in 2D space (Meszaros, 2022). I found a problem after using some methods because some points in this contig image have large values. In contrast, the values around it are minimal (x -axis 1118), which cannot be considered a red region. However, peak detection gives priority to them, so the solution we came up with was to convert the 2D Array to 1D by converting all The solution we came up with is to convert the 2D Array to 1D, averaging all the values on.




```
print(scaffold_1465[36,1118],scaffold_1465[15,659])
```

```
13 7
```

Reference List

Meszaros, A. (2022). Peak detection in a 2D Array. Retrieved 4 September 2022, from https://www.baeldung.com/cs/peak-detection-2d

Dekker, J. (2008). Gene Regulation in the Third Dimension. Science, 319(5871), 1793-1794. doi:10.1126/science.1152850

He, J., Li, C., Ye, B., & Zhong, W. (2012). Efficient and accurate greedy search methods for mining functional modules in protein interaction networks. *BMC Bioinformatics*, *13*(S10). doi: 10.1186/1471-2105-13-s10-s19

Matthey-Doret, C., Baudry, L., Breuer, A., Montagne, R., Guiglielmoni, N., & Scolari, V. et al. (2020). Computer vision for pattern detection in chromosome contact maps. Nature Communications, 11(1). doi: 10.1038/s41467-020-19562-7

Yoon, S., Chandra, A., & Vahedi, G. (2022). Stripenn detects architectural stripes from chromatin conformation data using computer vision. *Nature Communications*, *13*(1). doi: 10.1038/s41467-022-29258-9

H.JaseemaYasmin, J., & Mohamed Sadiq, M. (2012). An Improved Iterative Segmentation Algorithm using Canny Edge Detector with Iterative Median Filter for Skin Lesion Border Detection. *International Journal Of Computer Applications*, *50*(6), 37-42. doi: 10.5120/7779-0865

Wu, C. (2022). Retrieved 20 August 2022, from https://github.com/PlantandFoodResearch/HiC-heatmap-processing/blob/main/Screen%20Shot%202020-10-06%20at%2010.28.24%20AM%20(1).png