Sequence-Based Synteny Analysis of Multiple Large Genomes

Daniel Doerr and Bernard M.E. Moret

School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

Summary. Current methods for synteny analysis provide only limited support to study large genomes at the sequence level. In this chapter, we describe a pipeline based on existing tools that, applied in a suitable fashion, enables synteny analysis of large genomic datasets. We give a hands-on description of each step of the pipeline using four avian genomes for data. We also provide integration scripts that simplify the conversion and setup of data between the different tools in the pipeline.

Keywords: genome comparison, synteny analysis, marker sequences, large genomes

1 Introduction

The structural organization of genomes was studied long before the advent of genome sequencing technology, using chromosomal banding and other means, eventually leading to detailed chromosomal maps. Renwick [8] introduced the term *synteny* to denote close collocation of genomic elements on the same chromosome. With the advent of large-scale sequencing, synteny came to denote conserved collocation of genomic elements within large blocks of sequence and these blocks in turn became known as *syntenic blocks*.

Comparing the complete DNA sequences of large eukaryotic genomes for several species presents many challenges. The complexity of the necessary computations severely restricts the number of species and, even for three or four genomes, forces an undesirable trade-off between speed and accuracy. The large range of evolutionary changes, from sequence indels and point mutations through genome rearrangements and segmental duplications to deletions and insertions of entire blocks of sequence, serves to destroy collinearity and otherwise disguise similarities, thereby introducing ambiguities in the comparison. (This particular issue limits all syntenic analyses to collections of reasonably related genomes.) Multiway comparisons scale poorly, as the number of choices to be considered grows exponentially with the number of genomes to be compared. Finally, the genomes themselves are neither well defined (as they vary from one individual to the other within a population) nor perfectly sequenced and assembled (for instance, very few genomes are fully assembled, being given instead as contigs, with many poorly assembled genomes described with 10–100 times as many contigs as they have chromosomes), so that comparisons at the sequence level on a nucleotide-by-nucleotide basis are subject to many possible confounding factors.

The response to these challenges has been to compute a simplified representation of the genomes in terms of syntenic blocks and then carry out the multiway comparison at the level of syntenic blocks rather than at the level of base pairs. At first, researchers focused on genes as the base units, leading to studies of the gene content and gene order of genomes and mechanisms that affect these two representations. However, the vast majority of nucleotide bases in large genomes are not associated with genes, yet conservation across species is a powerful tool to identify functional elements and conduct meaningful comparisons among non-genic regions. Thus the focus shifted from genes to *genomic markers*, DNA segments that are well conserved across all or most of the given genomes. If each genome is found to have one (or more) contiguous region that contains the same markers (in the same order or in different orders), then each such region is a syntenic block and together these regions define a *syntenic block family*. (The similarity in terminology with genes and gene families is intentional and justified: genes are just very large markers.) The purpose of a synteny analysis is to produce families of syntenic blocks such that the blocks themselves are well conserved (have many shared markers and limited sequence variation overall) and the syntenic block families together cover much of each genome.

Syntenic blocks reduce the number of individual items to compare (in many analyses the length of these blocks varies from hundreds of thousands to as many as tens of millions of base pairs), afford some tolerance against missing or erroneous sequence data (from read calling errors to assembly errors) and against individual variation, and reduce the number of evolutionary events that must be considered when conducting a comparison (sequence-level events are no longer directly relevant), thus making comparison of multiple large genomes possible.

Because syntenic blocks and their families are just another representation of a collection of genomes, one would want to produce these blocks directly from sequence data, with as little prior knowledge as possible, and also produce blocks of various characteristics (coarser and larger blocks for quick analyses, fine-grained decompositions for accuracy and coverage, etc.). These two goals remain elusive, although much work is in progress for direct synteny analysis on unannotated genome sequences: tools such as Sibelia [6] and Satsuma [3] already provide solutions for bacterial genomes (Sibelia) or small datasets (Satsuma), but cannot handle a collection of large eukaryotic genomes.

In this chapter, we present a pipeline for synteny analysis of collections of large genomes. Our pipeline is based on several available tools. We provide scripts and give a hands-on description for each step of the pipeline, using four birds: *Gallus gallus* (chicken), *Meleagris gallopavoi* (turkey), *Taeniopygia guttata* (zebra finch), and *Ficedula albicollis* (collared flycatcher). Our scripts are available for download on GitHub[1]. Our pipeline uses the whole-genome alignment tool Mauve [1] and the synteny detection tool i-ADHoRe [7]. We also use Circos [5] for creating a genome-wide visualization of syntenic blocks. Our scripts are writ-

[1] https://github.com/danydoerr/large_syn_workflow

ten in Python 2 and require the Python library `Biopython` (and, optionally, `matplotlib`).

2 Preparing the genomic dataset

Careful selection of genome data is crucial to any synteny analysis. Many large genomes are only partially assembled, producing some modest number of contigs, alongside many short sequences of unknown or dubious origin. Such unassembled genomes pose some insurmountable difficulties to current synteny tools. While there exist other methods for scaffolding genomes based on comparative analyses [4], this task is not part of any synteny tool to date. (i-ADHoRe is able to identify syntenic blocks that are conserved across multiple contigs, but it only does so in the context of segmental or whole-genome duplication.) We therefore confine our workflow to fully or nearly assembled genomes.

As a first step, one should inspect the genome sequence files. First count the number and size of sequence records. Then, small or negligible sequences should be removed or assembled into larger sequences. The latter can be based on previous synteny analyses, comparative assembly, or simple sequential concatenation, so that in the end, each genome is represented by a manageable, clearly arranged set of sequences. Furthermore, each sequence should have a short but meaningful name by which it can be later identified when visually evaluating the outcome of the synteny discovery.

The analysis described in the following is based on genomic sequences retrieved from NCBI (https://ncbi.nlm.nih.gov/genome). We downloaded the following sequence files and computed their numbers of sequence records:

```
file name                                          #sequence records
GCA_000002315.3_Gallus_gallus-5.0_genomic.fna               23474
GCA_000146605.3_Turkey_5.0_genomic.fna                     231286
GCA_000151805.2_Taeniopygia_guttata-3.2.4_genomic.fna       37095
GCA_000247815.2_FicAlb1.5_genomic.fna                       21428
```

Each of these files contains thousands of sequence records with all but few of no relevance to our analysis, because they are too short. As a first step, we therefore discard all sequences that are smaller than 1Mbp. For this task, we provide a simple script called `extract_minlen_seq.py`, which takes as input a value for minimum sequence length and a FASTA file:

```
$ extract_minlen_seq.py 1000000 GCA_000002315.3_Gallus_gallus-\
    5.0_genomic.fna > C.min_1Mbp.fna
```

This task is repeated for the remaining sequence files, which are now labeled by a single character that identifies the corresponding species: C for chicken, T for turkey, Z for zebra finch, and F for collared flycatcher. The number of sequence records came down to a manageable few:

```
file name        #records  sequence record IDs
C.min_1Mbp.fna      31     CM000093.4..CM000107.4 CM000109.4.. \
                           CM000119.4 CM000121.4..CM000124.4 \
                           KQ759483.1
T.min_1Mbp.fna      30     CM000962.2..CM000978.2 CM000980.2.. \
                           CM000991.2 CM000993.2
Z.min_1Mbp.fna      39     CM000515.1..CM000532.1 CM000534.1.. \
                           CM000546.1 EQ832640.1 EQ832641.1 \
                           EQ832723.1 EQ832760.1 EQ832819.1 \
                           EQ832820.1 EQ833162.1 EQ833367.1
F.min_1Mbp.fna      39     CM001988.1..CM002017.1 CM002020.1 \
                           KE165308.1 KE165340.1..KE165342.1
```

With just a few exceptions, sequence records now correspond to chromosome scaffolds. Any remaining undesirable sequences can be individually removed from the sequence files.

Because the sequences have been downloaded from a public database, they are labeled with their database identifier. This label is necessary for bookkeeping, yet inconvenient for manual and visual analysis of the genomic dataset. Short labels, such as chromosome numbers, would simplify the manual analysis and provide short identifiers that can be used in the visualization. Because the FASTA files are too large to edit comfortably in a text editor and because, in some datasets, the preprocessing could involve the concatenation of some sequence records, we provide a script to edit FASTA files based on a limited set of instructions:

```
JOIN <ID 1> (h|t) <ID 2> (h|t) INTO <ID 3>
RENAME <ID 1> TO <ID 2>
REMOVE <ID>
```

The *rename* and *remove* operations are straightforward. The *join* operation requires, next to the sequence IDs, the extremities ("*h*" for *head* and "*t*" for *tail*) of the sequences that will be joined to produce a concatenated sequence with a specified ID. (In our example, concatenation is not necessary.)

Our script, `edit_sequences.py`, requires a sequence file and an instruction file, where each line corresponds to one edit operation, as shown by the contents of the edit file `C.edit` for the chicken dataset below:

```
-- file: C.edit ------------------------------------------
RENAME CM000093.4 TO 1
RENAME CM000094.4 TO 2
RENAME CM000095.4 TO 3
RENAME CM000096.4 TO 4
RENAME CM000097.4 TO 5
RENAME CM000098.4 TO 6
RENAME CM000099.4 TO 7
RENAME CM000100.4 TO 8
```

species	ID	version	scfs.	size (Gbp)	CDSs	markers	%cov.
Gallus gallus (chicken)	C	5.0	30	1.02	46,393	643,043	40.4
Ficedula albicollis (collared flycatcher)	F	1.5	30	1.04	26,464	643,043	39.5
Meleagris gallopavoi (turkey)	T	5.0	30	0.97	26,423	643,043	42.3
Taeniopygia guttata (zebra finch)	Z	3.2.4	31	1.02	19,447	643,043	40.3

Table 1. Genomic dataset of the four birds. Columns from left to right: Species name and parenthesized colloquial name; dataset version in NCBI; number of scaffolds after editing; genome size after editing; number of coding sequences (CDSs); number of markers identified by Mauve that are larger than 300bp and have homologs in all four birds; the percentage of nucleotide bases covered by these markers in their respective genomes.

```
RENAME CM000101.4 TO 9
RENAME CM000102.4 TO 10
RENAME CM000103.4 TO 11
RENAME CM000104.4 TO 12
RENAME CM000105.4 TO 13
RENAME CM000106.4 TO 14
RENAME CM000107.4 TO 15
RENAME CM000109.4 TO 17
RENAME CM000110.4 TO 18
RENAME CM000111.4 TO 19
RENAME CM000112.4 TO 20
RENAME CM000113.4 TO 21
RENAME CM000114.4 TO 22
RENAME CM000115.4 TO 23
RENAME CM000116.4 TO 24
RENAME CM000124.4 TO 25
RENAME CM000117.4 TO 26
RENAME CM000118.4 TO 27
RENAME CM000119.4 TO 28
RENAME CM000123.4 TO 33
REMOVE KQ759483.1
RENAME CM000121.4 TO W
RENAME CM000122.4 TO Z
-----------------------------------------------------------------
```

The final sequence files are obtained by calling `edit_sequences.py`:

```
$ edit_sequences.py C.min_1Mbp.fasta C.edit
```

Now, the sequence data of each genome in the dataset is purged from irrelevant content. Each dataset yields 30-31 sequence records, with 0.97Gbp to 1.04Gbp, as shown in Table 1.

3 Constructing genomic markers

We now provide a brief description on how to use the software tool progressiveMauve[2] [1] to obtain a genomic marker set across multiple genome sequences. Further information can be found in Mauve's comprehensive online user guide[3] and in the archive of the active Mauve user mailing list[4]. progressiveMauve is a multiple genome alignment tool which exploits unique local alignments in the genomic dataset as anchor blocks for a multiple sequence alignment. Using a guide tree, the method progressively aligns genomic sequences, using a sum-of-pairs anchor score.

A decisive advantage of progressiveMauve in comparison with competing methods is its uncomplicated use. In most analyses, no adjustments to its default parameter settings are necessary. Should customization be needed, it usually suffices to adjust a single parameter, the *match seed weight* (command line parameter `--seed-weight`). The match seed weight defines a threshold that alignment scores of unique matches must surpass in order to be considered in the initial construction of anchor blocks. Lower values will increase the sensitivity of the alignment and thus permit the validation of a reasonable number of markers when analyzing divergent genomes. Adjusting this parameter also influences alignment blocks that are subsequently identified through positional conservation, because, by default, the *minimum locally collinear block weight* (command line parameter `--weight`) is set to three times the match seed weight. Without changing any default settings, we call the program as follows:

```
progressiveMauve --output-guide-tree=C_F_T_Z.tree \
    --backbone-output=C_F_T_Z.backbone --output=C_F_T_Z.xmfa \
    C.fna F.fna T.fna Z.fna 2>&1 | tee mauve.log
```

Next to the alignment guide tree and the *XMFA* file, which will store the whole-genome alignment data, Mauve also provides a *backbone* table specifying which genomic segments have been matched with each other in the alignment. We will use the backbone table to extract the genomic markers and their affiliations for the synteny analysis in the following section.

4 Synteny analysis with i-ADHoRe

Most synteny analysis tools to date are only applicable to small genome datasets. i-ADHoRe [7] is one of the few tools that have been specifically designed to handle large eukaryotic genomes. It provides both fast algorithms and multithreading for the most time-consuming steps. The tool's configuration is defined through an input file containing the user's parameter settings. A comprehensive list of parameters and their options is provided in the manual included in the

[2] Available for download at http://darlinglab.org/mauve/download.html
[3] http://darlinglab.org/mauve/user-guide
[4] https://sourceforge.net/p/mauve/mailman/mauve-users

software package. Further, i-ADHoRe requires a proprietary format for its input files. Each contig/chromosome of any input genome must be represented by an individual file. For convenience, such files should be grouped in separate folders corresponding to their genome memberships. Relationships between markers must be provided in a further file, in one of two possible formats: either markers are associated with a family identifier (`table_type=family`) or pairwise relationships are individually provided, permitting also non-transitive relationships between markers (`table_type=blast`).

i-ADHoRe can be run in three different modes (parameter name `cluster_type`): `colinear`, `cloud`, and `hybrid`. The first mode detects collinear syntenic blocks in an iterative procedure, integrating marker content and order information from previous alignments into profiles for identifying further, more degenerate syntenic blocks. In doing so, each iteration leads to a higher level of syntenic block families with increasing size. The cloud mode intends to detect syntenic blocks that underwent internal rearrangements and so are better identified by their shared marker content than by ordering. How much the ordering can deviate from collinearity is bounded by a parameter (`q_value`). Lastly, the hybrid mode discovers collinear regions prior to performing synteny analysis in the cloud mode. Neither cloud nor hybrid mode supports the profile search performed in the collinear mode. For this reason, it is advisable to use the collinear mode due to its superior ability to identify degenerate syntenic blocks. In all three modes, i-ADHoRe allows gaps in syntenic blocks. Various parameters allow to fine-tune the size of these gaps in different contexts. In practice, it often suffices to control their setting through a shared value.

Processing the genome data and Mauve's backbone table into the input format required by i-ADHoRe is an elaborate task. Therefore, we provide a script which performs this conversion conveniently in one program call. Our script, called `mauve2iadhore.py`, offers a few parameter settings to adjust the most influential parameters of i-ADHoRe directly upon creating the input files. It can also filter segments of Mauve's backbone file that fall below a given minimum length. All options are listed in Table 2. Our script will also enable i-ADHoRe's multithreading feature by providing it with the number of cores that are available on the machine on which the script is executed.

For the analysis of the four avian genomes, we call `mauve2iadhore.py` as shown below. We require that segments in Mauve's backbone file that will be used as markers in subsequent analysis must be conserved across all four genomes (`-n4`) and have a minimum length of 300bp (`-l300`). In our bird dataset, Mauve identifies many markers that fit our constraints (see Table 1), many more than the number of coding sequences in these genomes, enabling a more fine-grained synteny analysis. The price to pay: the gap size for syntenic blocks in the subsequent i-ADHoRe run must be set appropriately high (`-g300`).

```
$ mauve2iadhore.py -c colinear -n4 -l300 -g300 C_F_T_Z.backbone \
    C.fna F.fna T.fna Z.fna
```

The script produces then the following files and folders:

```
C/  C_F_T_Z.log  F/  T/  Z/  blast_table.txt  dataset.ini

./C: 1.lst 10.lst 11.lst 12.lst 13.lst 14.lst 15.lst 17.lst \
    18.lst 19.lst 2.lst 20.lst 21.lst 22.lst 23.lst 24.lst \
    25.lst 26.lst 27.lst 28.lst 3.lst 4.lst 5.lst 6.lst 7.lst \
    8.lst 9.lst W.lst Z.lst

./F: 1.lst 10.lst 11.lst 12.lst 13.lst 14.lst 15.lst 17.lst \
    18.lst 19.lst 1A.lst 2.lst 20.lst 21.lst 22.lst 23.lst \
    24.lst 25.lst 26.lst 27.lst 28.lst 3.lst 4.lst 4A.lst \
    5.lst 6.lst 7.lst 8.lst 9.lst Z.lst

./T: 1.lst 10.lst 11.lst 12.lst 13.lst 14.lst 15.lst 16.lst \
    17.lst 19.lst 2.lst 20.lst 21.lst 22.lst 23.lst 24.lst \
    25.lst 26.lst 27.lst 28.lst 29.lst 3.lst 30.lst 4.lst \
    5.lst 6.lst 7.lst 8.lst 9.lst Z.lst

./Z: 1.lst 10.lst 11.lst 12.lst 13.lst 14.lst 15.lst 17.lst \
    18.lst 19.lst 1A.lst 1B.lst 2.lst 20.lst 21.lst 22.lst \
    23.lst 24.lst 25.lst 26.lst 27.lst 28.lst 3.lst 4.lst \
    4A.lst 5.lst 6.lst 7.lst 8.lst 9.lst Z.lst
```

Finally, we call i-ADHoRe by passing it the created configuration file `dataset.ini`:

```
$ i-adhore dataset.ini
```

According to the configuration set up by our conversion script, i-ADHoRe deposits its output files in a folder named `output` below the current working directory. Among the various files that i-ADHoRe creates, `output/multiplicons.txt` and `output/segments.txt` contain the central information regarding the discovered syntenic block families.

5 Evaluation

We evaluate syntenic blocks based on synteny criteria suggested by Ghiurcuta and Moret [2] and provide a genome-wide visualization of synteny blocks between pairs of species of our dataset.

Ghiurcuta and Moret presented two measures for scoring a syntenic block family: the *relaxed score* represents the fraction of markers that are connected to any other markers within their syntenic block family; conversely, the *weighted score* is a stricter measure describing the fraction of markers that are connected to at least one marker in each syntenic block.

We provide a script, `synteny_score.py` that computes these scores from the output of an i-ADHoRe run. Our script prints for each syntenic block family its corresponding relaxed or weighted synteny score. Optionally, it is also able to

Options	Default value	Description
`-n INT, --quorum=INT`	2	Minimum number of species in which each marker must have a homologous counterpart.
`-l INT, --minlength=INT`	100	Minimum length a segment must have to be included in the output.
`-g INT, --max_gap_size=INT`	30	Maximum distance between markers in a cloud or collinear cluster.
`-c CLUSTERTYPE, --cluster_type=CLUSTERTYPE`	colinear	i-ADHoRe cluster type, must be any of (colinear | hybrid | cloud)
`-q [0, 1], --q_value=[0, 1]`	0.5	Minimum r^2 distance for markers in a cloud cluster.
`-o OUTDIR, --outdir=OUTDIR`	"."	Directory to which the output files will be written.

Table 2. Command line options of conversion script `mauve2iadhore.py`.

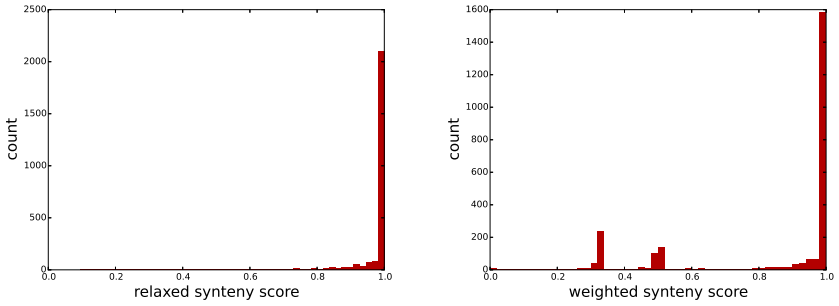

Fig. 1. Histogram of relaxed and weighted scores of syntenic blocks in the avian dataset identified by i-ADHoRe.

directly create a plot that visualizes a histogram over all synteny scores. To this end, the script requires the python package `matplotlib` library, which must be installed on the system.

We can now evaluate the syntenic block families discovered by i-ADHoRe in the avian dataset. To visualize the histograms shown in Figure 1, we call our script by providing the `-v` parameter as follows:

```
$ synteny_scores.py -t relaxed -v dataset.ini output/segments.txt
$ synteny_scores.py -t weighted -v dataset.ini output/segments.txt
```

By default, the calls create two files containing the figures, `relaxed_scores.eps` and `weighted_scores.eps`, respectively, which will be located in the current working directory. However, an alternative output file name can be specified with the `-f` parameter.

Although i-ADHoRe provides its own graphical output, it does not produce plots to study the entire set of syntenic blocks across whole genomes. Therefore, we created an alternative genome-wide visualization, using the popular visualization tool Circos [5]. For each pair of genomes, we instruct Circos to draw all their synteny blocks and relationships. Our visualization permits a quick assessment of the segmentation of genomes into markers and syntenic blocks, of gene order divergence, and of genome coverage.

We call script `iadhore2circos.py` on the avian dataset as follows:

```
$ iadhore2circos.py -c dataset.ini output/multiplicons.txt \
    output/segments.txt
```

The script creates for each genome pair an individual *Circos configuration file*, but also further configuration files such as those containing karyotype information for the dataset. The `-c` parameter causes each link between syntenic blocks to be drawn in a different color. Otherwise, all links associated with the same chromosome are drawn in the same color. Further, our script provides the option `-l` to draw higher-level intervals identified by i-ADHoRe in `colinear` mode. It remains to call the Circos binary on each of these files, as exemplified below for the genome pair chicken and collared flycatcher.

```
$ circos -conf C_F.circos.conf
```

The plots thus created are shown in Figure 2. In each plot, the outer circle shows the pair of genomes drawn in blue and green, respectively, partitioned into its contigs/chromosomes. Black bars drawn on top correspond to locations of markers. Lastly, as mentioned above, colored links indicate relations between syntenic blocks among the two species.

References

1. Darling, A.E., Mau, B., Perna, N.T.: progressiveMauve: Multiple genome alignment with gene gain, loss and rearrangement. PloS ONE 5(6), e11147 (2010)
2. Ghiurcuta, C.G., Moret, B.M.E.: Evaluating synteny for improved comparative studies. Bioinformatics 30(12), i9–18 (Jun 2014)
3. Grabherr, M.G., Russell, P., Meyer, M., Mauceli, E., Alföldi, J., Di Palma, F., Lindblad-Toh, K.: Genome-wide synteny through highly sensitive sequence alignment: Satsuma. Bioinformatics 26(9), 1145–1151 (May 2010)
4. Husemann, P., Stoye, J.: r2cat: synteny plots and comparative assembly. Bioinformatics 26(4), 570–571 (Feb 2010)
5. Krzywinski, M.I., Schein, J.E., Birol, I., Connors, J., Gascoyne, R., Horsman, D., Jones, S.J., Marra, M.A.: Circos: An information aesthetic for comparative genomics. Genome Research 19(9), 1639–1645 (Jun 2009)
6. Minkin, I., Patel, A., Kolmogorov, M., Vyahhi, N., Pham, S.: Sibelia: A Scalable and Comprehensive Synteny Block Generation Tool for Closely Related Microbial Genomes. In: Theory and Applications of Models of Computation, pp. 215–229. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

7. Proost, S., Fostier, J., De Witte, D., Dhoedt, B., Demeester, P., Van de Peer, Y., Vandepoele, K.: i-ADHoRe 3.0–fast and sensitive detection of genomic homology in extremely large data sets. Nucleic Acids Res. 40(2), e11–e11 (Jan 2012)
8. Renwick, J.H.: The mapping of human chromosomes. Annu. Rev. Genet. 5(1), 81–120 (1971)

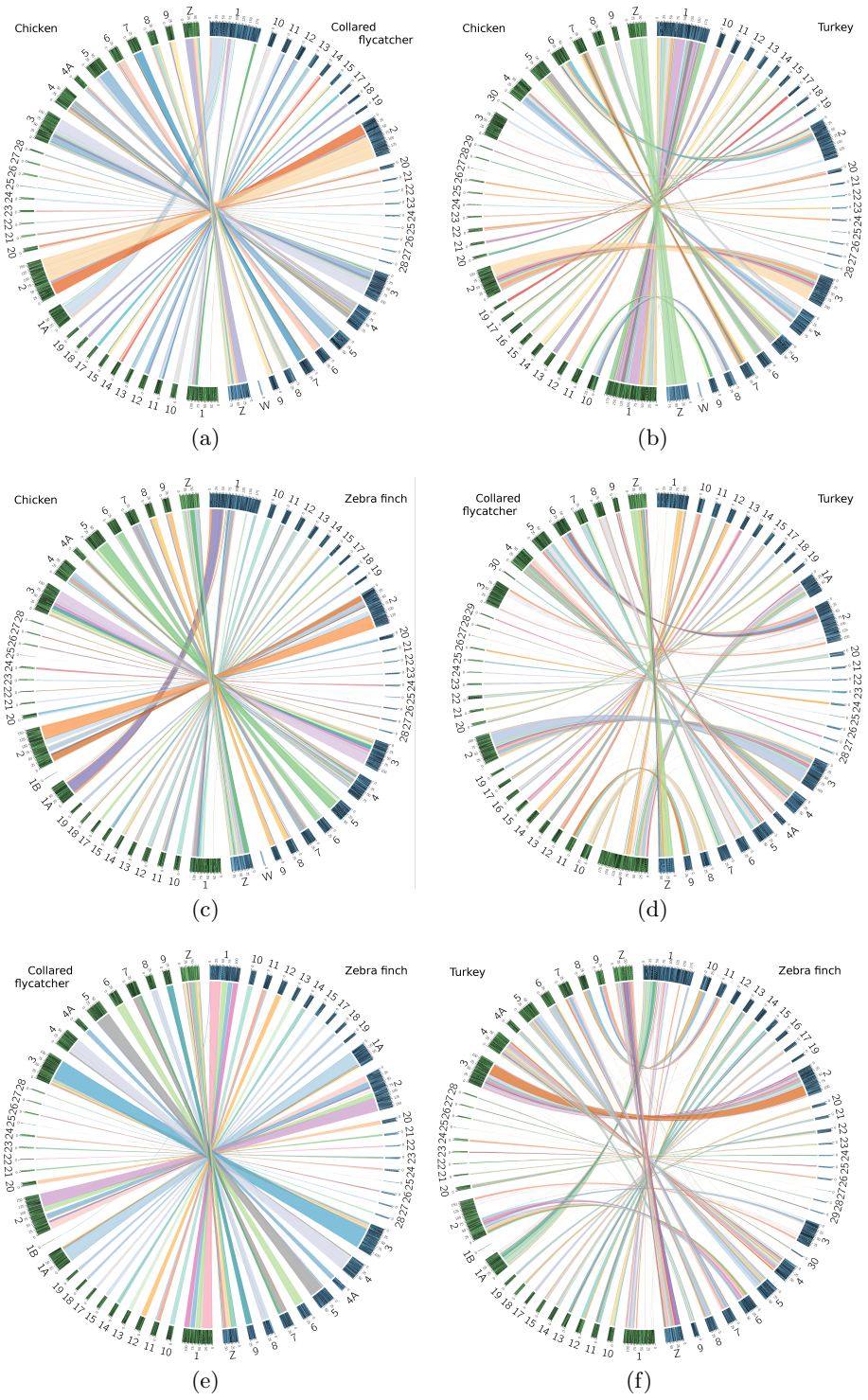

Fig. 2. Circos plots visualizing the pairwise comparison of syntenic blocks in four avian genomes.