AccuSyn: Using Simulated Annealing to Declutter Genome Visualizations

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jorge Dionisio Nunez Siri

Permission to Use

Abstract

We apply Simulated Annealing, a well-known metaheuristic for obtaining near-optimal solutions to optimization problems, to discover conserved synteny relations (similar features) in genomes. The analysis of synteny gives biologists insights into the evolutionary history of species and the functional relationships between genes. However, as even simple organisms have huge numbers of genomic features, syntenic plots initially present an enormous clutter of connections, making the structure difficult to understand. We address this problem by using Simulated Annealing to minimize link crossings. Our interactive web-based synteny browser, AccuSyn, visualizes syntenic relations with circular plots of chromosomes and draws links between similar blocks of genes. It also brings together a huge amount of genomic data by integrating an adjacent view and additional tracks, to visualize the details of the blocks and accompanying genomic data, respectively. Our work shows multiple ways to manually declutter a synteny plot and then thoroughly explains how we integrated Simulated Annealing, along with human interventions as a human-in-the-loop approach, to achieve an accurate representation of conserved synteny relations for any genome. The goal of AccuSyn was to make a fairly complete tool combining ideas from four major areas: genetics, information visualization, heuristic search, and human-in-the-loop. Our results contribute to a better understanding of synteny plots and show the potential that decluttering algorithms have for syntenic analysis, adding more clues for evolutionary development. At this writing, AccuSyn is already actively used in the research being done at the University of Saskatchewan and has already produced a visualization of the recently-sequenced Wheat genome.

Acknowledgements

I would like to deeply thanks my supervisor, Dr. Eric Neufeld, for being the best supervisor one can ever ask for. I feel more than blessed for having him throughout these two years of research, where he was completely involved in every aspect of my academic life. Thank you, Eric, for your generous support and for believing in me since the first day. Your continuous guidance, commitment, and inspiration allowed this thesis to be my own work, having you leading me every time I needed it. Thank you for your patience and all the time you dedicated to countless meetings, thesis edits, and careful explanations. I will always remember our weekly meetings, where we could sit for hours to have research discussions and even continue them through many email threads. All this thesis work would not have been possible without you.

Besides the generous funding I received from Eric, I am also grateful for the scholarship offered by the Department of Computer Science at the University of Saskatchewan, which made all this research possible. Thank you for also allowing me the opportunity to help faculty members and students through teaching assistantships.

I would like to give my special thanks to Dr. Andrew Sharpe, Dr. Isobel Parkin, Dr. Carl Gutwin, Dr. Debajyoti Mondal, Dr. Sean Walkowiak, Kevin Koh, and Venkat Bandi, for offering their thoughtful feedback to make this work better and for their help in configuring the datasets. Also, I would like to thanks my thesis committee, Dr. Ian McQuillan and Dr. Michael Horsch, whose comments and suggestions have improved this thesis.

I would like to express my heartfelt gratitude to Gwen Lancaster, for all your support at the beginning of my Master's degree and for helping both my wife and me on every aspect, so that we could successfully settle in Canada. Thank you to all the staff members of the Department of Computer Science that helped me along the way, in particular, Sophie Findlay, Heather Webb, and Shakiba Jalal. Also, thank you to my friends at the IMG lab, for your support and for coming to my thesis defense.

Last but not least, I am particularly grateful for the exceptional support given by my family and my wife. Thank you to my parents and my brothers, for being there for me despite the distance and for always being proud of my success. Your motivation and your love kept me focused on my end goal. Thank you to my wife, Rubi Lopez, for supporting me every day and for getting so involved in my research since the beginning. Thank you for your patience and your love that allowed me to be the best version of myself. This is an achievement for both of us and with you, I feel that all the effort was worth it.

Thank you all for being part of this thesis and helping me grow academically, professionally, and personally.

Jorge Dionisio Nunez Siri

This thesis is dedicated to my wife and my family – for all their unconditional love and support.

Contents

List of Tables

List of Figures

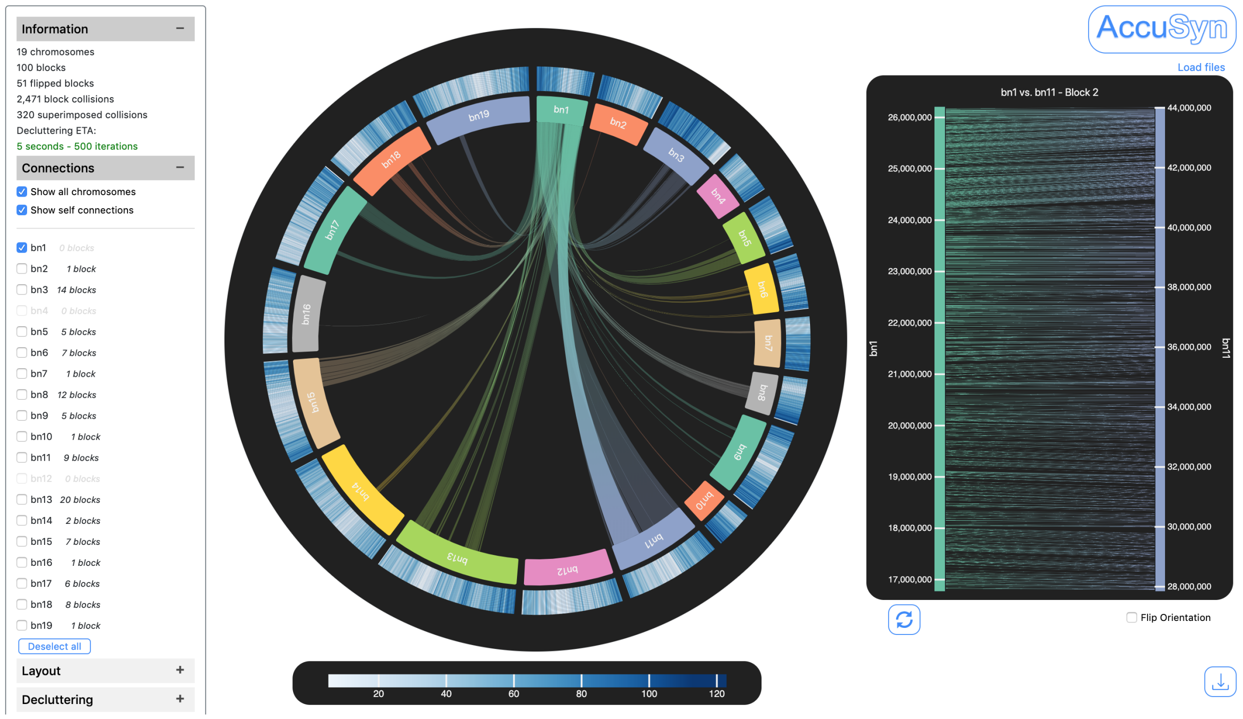
List of Abbreviations

BLAST	Basic Local Alignment Search Tool
CSS	Cascading Style Sheets
DNA	Deoxyribonucleic Acid
ETA	Estimated Time of Arrival
FASTA	Fast All
GFF	Genomic Feature File
HTML	Hypertext Markup Language
JS	JavaScript
JPEG	Joint Photographic Experts Group
MCScan	Multiple Collinearity Scan
NP-complete	Nondeterministic Polynomial time complete
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics

1 Introduction

AccuSyn is an accurate web-based genome synteny browser created using D3.js [5] to make efficient genomic data exploration with smooth transitions possible. It visualizes syntenic relations among chromosomes, where synteny means related long DNA sequences that are very similar, grouped into blocks to make their comparison easier. AccuSyn provides a high-level of interactivity and performance, letting users obtain meaningful insights when exploring genomic data in two side-by-side views: the genome and the block view.

The genome view uses a circular plot to allow the exploration of block relationships between chromosomes, either by visualizing links from the blocks in one chromosome to all the others, or all the many-to-many connections between chromosomes. It uses CircosJS [18], a library based on D3.js [5] for visualizing data in Circos plots [44]. The block view uses a bipartite bar plot to show all the pairs of gene locations for any block, by using separate scales for each chromosome.


Information

19 chromosomes
100 blocks
51 flipped blocks
2,471 block collisions
320 superimposed collisions
Decluttering ETA:
5 seconds - 500 iterations

Connections

☑ Show all chromosomes
☑ Show self connections

☑ bn1	0 blocks
☐ bn2	1 block
☐ bn3	14 blocks
☐ bn4	0 blocks
☐ bn5	5 blocks
☐ bn6	7 blocks
☐ bn7	1 block
☐ bn8	12 blocks
☐ bn9	5 blocks
☐ bn10	1 block
☐ bn11	9 blocks
☐ bn12	0 blocks
☐ bn13	20 blocks
☐ bn14	2 blocks
☐ bn15	7 blocks
☐ bn16	1 block
☐ bn17	6 blocks
☐ bn18	8 blocks
☐ bn19	1 block

Deselect all
Layout +
Decluttering +

Figure 1.1: AccuSyn providing a complete interface overview using the *Brassica napus* genome. The genome view on the middle visualizes all the relationships from chromosome "bn1" to all the others. The block view on the right shows all the pairs of syntenic genes inside "block 2" between chromosomes "bn1" and "bn11." The additional track around the genome view shows the gene frequency. The legend in the bottom confirms the values of the continuous color scale in the additional track.

AccuSyn visualizes synteny using two files as input: a Genomic Feature Format (GFF) file that describes the genome coordinate information, and a collinearity file, with all the syntenic blocks of similar genes grouped together. The latter is the output of MCScanX, a popular tool for detecting synteny and collinearity [77]. It also supports BedGraph files to display additional data as tracks in the inner or outer portion of a genome.

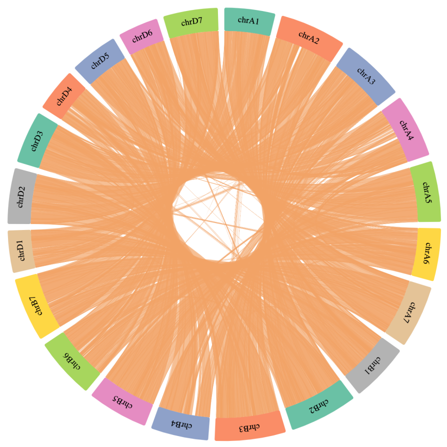
Recent studies show that, due to innovations in DNA sequencing technology, data analysis is replacing data generation as the rate-limiting step in genomic studies [51]. To enhance the analysis of conserved synteny data, the main contribution of this thesis is to automate the creation and detangling of synteny diagrams. To explain it in a summarized way we use Figure 1.2, which includes six images showing the decluttering and enhancement process of the Chinese Spring Wheat genome, a recently-sequenced plant genome [38]. Part (a) shows the initial figure with all the chromosomes selected, revealing that with one color the view is nearly impossible to declutter. Part (b) shows how just adding color to the blocks helps identify relationships. Part (c) starts to clean the view by filtering by block length, leaving only the denser blocks present. Adding color and filtering are two straightforward ways to start to recognize patterns.

However, the real decluttering happens when grouping the most related chromosomes together as shown in Figure 1.2(d). Before AccuSyn, detangling was often done manually, that is, the user had to manually input the ideal chromosome order. Therefore, we turned our attention to the automation of the display of syntenic sequences. We applied Simulated Annealing, a well-known heuristic for obtaining near-optimal solutions to optimization problems, to automate the process of discovering syntenic relations by minimizing link crossings in the genome view. Even though the view gets decluttered when having fewer block crossings, Figure 1.2(e) shows that after grouping chromosomes we can reverse the filtering to display all the relationships and see the precision of the detangling. Finally, Figure 1.2(f) reveals how easily we can include additional tracks and show genome attributes in dark mode to have publication-ready plots with even better contrast.

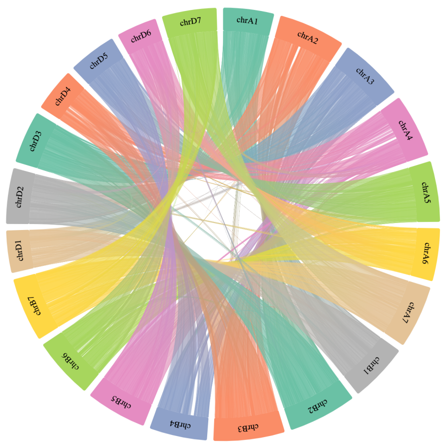
In this exploratory research, we were repeatedly getting requirements and seeing which results were worthwhile for users. Simulated Annealing turned out to be a productive first step in this search. What worked even better was applying Simulated Annealing along with human interventions. AccuSyn provides users with two interactive ways to declutter a genome: by dragging chromosomes around or flipping block locations. It also incorporates an interface for non-experts to interact with the algorithm multiple times while being able to adjust its parameters and go back and forth between runs.

To the best of our knowledge, our result is the first approach to automate the decluttering of a genome, while also allowing user interventions and visualizing a huge amount of genomic data all at once. Due to its interdisciplinary nature, all this work was by no means exhaustive. It required the integration of four areas of knowledge: genetics, information visualization, heuristic search, and human-in-the-loop.

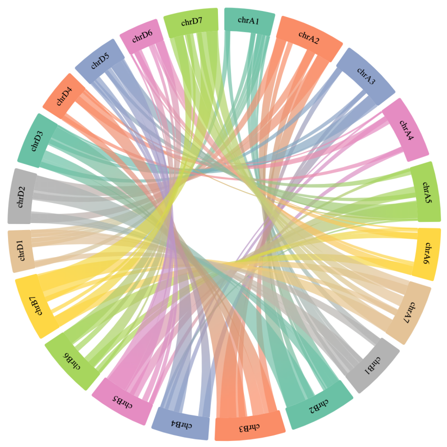
Next, we discuss the background of our work: the biological background, related synteny visualizations, dataset, the crossing number problem, and some examples of heuristic search algorithms and human-in-the-loop. We then present our proposed solution to declutter syntenic diagrams, followed by additional considerations and a discussion of our implementation. Finally, we present our conclusions and directions for future work.



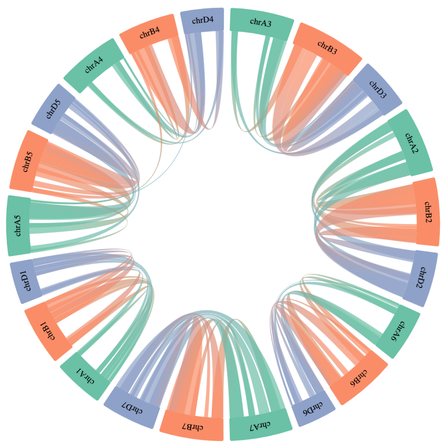
(a) One color.



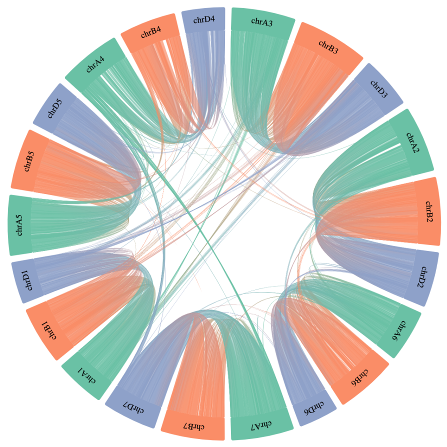
(b) Colored.



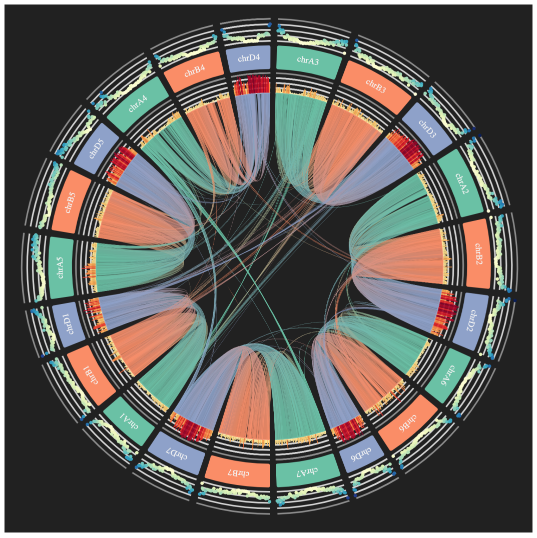
(c) Filtered.



(d) Grouped.



(e) Unfiltered.



(f) Complete.

Figure 1.2: AccuSyn providing a step-by-step decluttering of the *Chinese Spring Wheat* genome.

2 Background

2.1 Biological background

This biological background is intended for readers who need to understand the main biology to know how AccuSyn works. In comparative genomics, biologists study similarities between genomes by analyzing relationships between genomic features. A genome contains the complete genetic information of an organism. A genomic feature is any element of interest. In this case, we are interested in analyzing genes.

The genome of a species is composed of long chains of DNA called chromosomes, each of which contains hundreds of genes, the basic unit of heredity and biological information. Each gene consists of a sequence of nucleotides at specific positions on a particular chromosome. A gene expresses itself as a phenotype, defining the look or the behavior of an organism. DNA molecules are made up of two paired strands of nucleotides, each of which consists of a deoxyribose sugar, a nitrogenous base pair, and a phosphate group.

The four bases are two purines, adenine (A) and guanine (G), and two pyrimidines, thymine (T), and cytosine (C). Within the DNA molecule, bases on opposite strands pair specifically; A bonds only with T and C bonds only with G. These bonds make the double-stranded DNA structure and measure the size of an individual gene or a whole chromosome. For instance, the human genome is made up of an estimated three billion base pairs with about 24,000 distinct protein-coding genes [8]. To deal with such big numbers, biologists use measures such as kilobases (Kb) for 1,000 base pairs; megabases (Mb) for one million base pairs; and gigabases (Gb) for one billion base pairs.

Throughout evolution, chromosomes can change, or mutate, in certain ways through heritable changes in DNA base sequences, including substitutions, when a base at a certain position is replaced by one of the three other bases; deletions, when one or more nucleotide pairs are lost from a DNA molecule; insertions or duplications, when one or more nucleotide pairs are added to a DNA molecule; inversions, when a piece of chromosome breaks, rotates 180 degrees, then fuses back in place; translocations, when parts of two chromosomes break off and swap places [29]. These chromosomal rearrangements cause significant genetic reorganizations, resulting in loss or increase of synteny between species.

Another important mechanism for altering a DNA sequence involves transposable elements (TEs), also known as "jumping genes" or transposons. They are segments of DNA, sometimes hundreds to thousands of base pairs long, that either copy or move (depending on the type) from one location to another in a genome. When a TE jumps into a gene, it can cause a mutation that disrupts the gene's function. Transposable elements can move in two ways: by completely leaving their original position, or by copying themselves into a different chromosomal location. The latter mutations are known as retrotransposons because their DNA sequence gets repeated throughout the genome [29]. Transposons can also help biologists in finding patterns, i.e. gaining insights on the origin of certain genes. They can track the movements of the transposons and infer where genes might have come from, which is valuable when looking at the evolution of any species.

Due to the huge number of genes in many genomes, algorithms group genomic features into syntenic blocks, contiguous sets of genes located on the same chromosome. Conservation implies the relatedness or similarity of these blocks of genes within one or between two or more species [49]. Thus, a conserved gene is one that has not been significantly disrupted throughout evolution. The term "synteny," which literally means "on the same ribbon," refers to genes that are located on the same chromosome [49, 54]. Conserved synteny happens when the co-localization of related genes within genomic blocks is preserved among chromosomes, but not necessarily in the same order [57]. Collinear synteny or collinearity, a particular form of synteny, happens when two conserved regions ideally share the same order of genes [58, 77]. Hence, a genome synteny browser visualizes both the order of genes and the relationships of syntenic sequences.

Biologists use syntenic data to find evidence about the homology of species, which is defined as shared ancestry between two similar biological sequences (*genes*) [70]. One can understand sequence homology, by considering the following real-life scenario. If students in a classroom are asked to write down a random phrase on a piece of paper, there is a chance that one of them will start copying from the person beside them, meaning that this student does not know what to write, and thus, looks at the neighbor and writes the same phrase but perhaps changing a few words. If we extend this example to the point where multiple students do the same, then there is an evolutionary relatedness between the phrases. The same happens with genomes, where genes can mutate and change their sequences. In this case, these new instances are homologous to their common ancestor and can be correctly aligned by evolutionary descent.

Chromosomes that match in size and shape are called homologous chromosomes, or homologs. Diploid organisms, like humans, have two homologous copies of each chromosome, one genetic copy from each parent. For instance, humans have a total of 46 chromosomes, 22 pairs of homologs and one pair of sex chromosomes (X and Y) [8]. Moreover, organisms that contain more than two sets of homologs are considered polyploid. This polyploidy state is especially common in plants, like *Brassica napus*, considered an amphidiploid or allotetraploid. *Tetraploid* means that it has four copies of each homolog or two diploid sets from each parent, and the prefixes *allo* or *auto* are used when these copies come from different or same species, respectively. In fact, the *B. napus* genome has 19 chromosomes, derived from two diploid genomes, *Brassica rapa* with 10 chromosomes and *Brassica oleracea* with nine chromosomes. Therefore, *B. napus* is considered allotetraploid because its first 10 chromosomes are homologs of *B. rapa* and its last nine chromosomes are homologs of *B. oleracea* [7].

A polyploidy state has to do with finding insights into multiple genome copies when studying synteny diagrams. The duplicated regions found on the syntenic structure of genomes give evidence of ancestral polyploidy [78]. Furthermore, it is because of this polyploidy state that the majority of plants have huge numbers of duplicated genes. When looking at the ancestors of *B. napus*, for almost every locus detected in the *B. rapa* genome, a homologous locus can be found in the *B. oleracea* genome, having up to 73% of duplicated sequences, which explains the duplicated structure of the allotetraploid *B. napus* [53]. Similar insights can be found in the allohexaploid *Triticum aestivum* (Wheat genome) [38], that has three similar diploid subgenomes A, B, D, hence having a triplicated structure. Likewise, the allohexaploid *Camelina sativa* [40] also has a triplicated structure.

2.1.1 Local sequence alignment

Similarity searching is one of the major techniques used by computational biologists. Before looking at similarities between two sequences, their correct alignment must be computed. As a consequence of genome mutations and an artifact of any alignment, gaps are introduced into one or both sequences to produce an optimal alignment. There are two types of alignments: local sequence alignment finds an optimal match between any pair of subsequences and global sequence alignment uses the two entire sequences to find the best alignment. Local alignments are the most relevant to biological applications since they are used to compare smaller to larger regions, such as finding out conserved patterns in a DNA sequence when comparing it to an entire database.

The Smith-Waterman dynamic programming algorithm [66] is a correct and complete computational solution to perform the optimal local sequence alignment of two sequences. It uses a matrix and a scoring scheme to represent matches, mismatches, and gaps when comparing subsequences of all possible lengths. Smith-Waterman exhibits a quadratic behavior when optimizing the similarity measure, that is, $O(m \times n)$, where m and n are the lengths of the two sequences being aligned. This complexity time is not ideal to achieve an answer in a reasonable amount of time for very big sequences, considering that the size of sequence databases is exceeding the increases in computational speed [56].

To avoid this cost, heuristic methods are used for local sequence alignment. FASTA (*Fast-All*) [46, 55] was the first widely used heuristic algorithm for similarity searching in databases, i.e. aligning a query genome sequence to an entire database. Its main legacy is the FASTA file format, which is now universal in bioinformatics, representing the query sequence using a set of identifiers along with nucleotide or protein sequences as a series of single-letter characters.

BLAST (*Basic Local Alignment Search Tool*) [2, 56] is the most popular tool for local sequence alignment against a database. Similar to FASTA, it is a heuristic algorithm used to compare a query sequence to a reference database, searching for regions of local similarity. It is seen as the successor of FASTA since it is more time-efficient and adopts the FASTA file format for the input query sequences. There are two main variations, BLASTP and BLASTN, used to compare a protein or nucleotide sequence to a database, respectively. When comparing sequences, BLAST uses a scoring matrix to judge the quality of the alignment. Commonly used matrices include BLOSUM (*Blocks Substitution Matrix*) and PAM (*Point Accepted Mutation*) matrix, where each value describes the probability of a biologically meaningful matching pair occurring in an alignment.

The BLAST heuristic works in three steps: first, it generates a list of all the words (i.e. sequence of characters of fixed length w) from the query, expanding it to include matching words from the scoring matrix, and keeping only the top high-scoring matches; second, it searches through the target database for exact matches to the word list generated; third, for each match, the alignment is extended in both directions as long as the score increases above the score threshold. The results are called high-scoring pairs (*HSPs*), which are used to determine the statistical significance of each score or expectation value (*e-value*), i.e the number of alignments with the same or greater score that one is likely to see by chance when searching a database of similar size. In the end, BLAST is not guaranteed to find the best local alignment, but it is used as an accurate approximation.

2.1.2 Synteny identification

When doing biological sequence comparisons, BLAST uses sequence similarity to approximate matches from genome sequences. However, sequence similarity alone is not enough to help infer homology. Koski and Golding [42] found that even though it is a common practice to state the most significant BLAST hit as a result in a comparative genomics report, this hit is often not the nearest neighbor, i.e. the most similar BLAST result is not necessarily the closest in the evolutionary tree. This is a statement about homology and hence one should be careful when using BLAST to infer it.

While BLAST does a pair-wise comparison between a gene and all the gene sequences in a database (or some organism of interest), synteny tools generally use the alignment of genes paired by BLAST along with genomic location coordinates to create syntenic blocks of genes. Therefore, synteny tools provide an extra layer of information to confirm gene homology, being more reliable than sequence similarity results alone [70]. Moreover, synteny detection is important for two use cases: looking for patterns on functional relationships between similar genes, as conserved syntenic genes are more likely to retain the same expression pattern, or tracing location changes of genes across related species, i.e. how a genome has evolved over time [70, 77]. This enables straightforward conversion from synteny information to getting comprehensive insights that are able to inform any further downstream analysis.

One of the project requirements for AccuSyn was to use MCScanX to create and detect syntenic blocks. MCScanX [77] scans multiple genomes or subgenomes to identify homologous chromosomal block regions and align them using genes as anchors. MCScanX generates multiple alignments of collinear blocks by using a simplified Genomic Feature Format (GFF) file (described later in this chapter) and a BLASTP file as inputs, which contains all the protein sequences of the species of interest, used to compute collinear blocks of genes for every possible pair of chromosomes. The MCScanX algorithm is an adjusted version of MCScan (*Multiple Collinearity Scan*) [71]. It works by first sorting BLASTP gene matches according to chromosomal positions with the help of the GFF file. Then, it uses dynamic programming to find the most significant chains of collinear gene pairs (i.e. duplicated block regions with conserved gene order) having a scoring schema that rewards and penalizes adjacent and distanced gene pairs, respectively. Finally, MCScanX reports only the blocks involving at least five collinear gene pairs.

Wang et al. [77] state that detection of synteny is complicated because of gene loss, tandem duplications (i.e. groups of clustered homologous genes in a genome), gene transpositions, and chromosomal rearrangements since they can produce experimental biases (*artifacts*). Synteny detection tools can be classified in two ways when looking at how they work [77]. First, tools like i-ADHoRe (*iterative Automatic Detection of Homologous Regions*) [58] and OrthoCluster [79] identify synteny by the clustering of neighboring similar gene pairs. The second class of tools includes, among others, DAGchainer [25], SyMAP [67], SynFind [70], and MCScanX [77]. They use a more advanced strategy by applying *dynamic programming* to duplicated regions with conserved gene order and assigning a score to each resulting block taking into account the similarity and closeness of each gene pair.

Dynamic programming works by first partitioning a problem into overlapping sub-problems until achieving a trivial solution. Then, it uses those unique solutions to construct solutions for larger pieces of the original problem, avoiding the calculation of the same sub-problem twice [56].

Besides having algorithmic differences, synteny tools usually differ in their required input files, output format, and computational cost and efficiency. Even though AccuSyn has been created to accept MCScanX file format (described later in this chapter), we do not rule out that the use of any other synteny detection tool may give good results.

2.2 Related synteny visualizations

This section explains some of the latest attempts to visualize synteny. Visualizing syntenic relationships gives biologists the ability to quickly understand and confirm detected syntenic patterns to provide insights into the evolutionary history of a species, which in turn informs plant breeders.

Synteny detection tools often include basic synteny visualizations that help in making downstream analysis easier. For instance, MCScanX [77] displays its results using the four most common types of plots. Linear and circular plots visualize genomes arranged in lines or around a circle, respectively, using connected ribbons to represent the syntenic blocks. Dot plots visualize syntenic relations in a scatterplot by representing each genome as an axis and the syntenic blocks as dots in the plot. Lastly, bar plots visualize genomes using bars for each chromosome and representing syntenic blocks with colored regions instead of connections. However, when bundled inside synteny identification tools, these visualizations are often limited and not interactive, creating the need for more complete visualization tools to assist researchers in finding syntenic patterns straightforwardly.

Circos [44] is a command-line non-interactive viewer that uses a circular layout to display relationships between genomic positions using ribbons (as shown in Figure 2.1). It leverages the huge amount of genomic data by displaying additional data in multiple tracks, such as heatmaps, histograms, lines, and scatter plots.

Therefore, its main purpose is to reduce the implicit difficulties when visualizing large-scale genomic data. Circos takes as input files similar to the Genome Feature Format (GFF) file format and outputs images in PNG or SVG formats. Nowadays, Circos plots are often the norm when providing new synteny results in the field of comparative genomics, e.g. [38, 40]. Circos is the first viewer that uses circular plots to visualize similar relationships, making it ideal to get a complete overview of a genome. As the authors described it, Circos presents "images that are clear and informative to the investigator and attractive and compelling to the general public" [44].

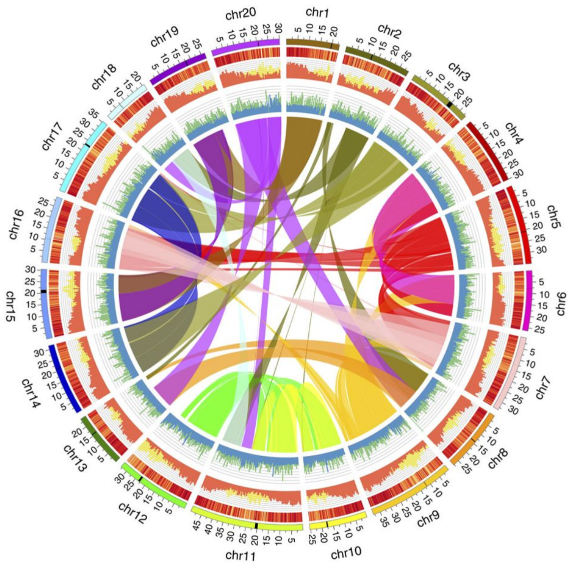


Figure 2.1: Example of the Circos viewer [44] showing the Camelina sativa plot. Extracted from *Kagale et al.* [40].

MizBee [49] is a stand-alone synteny browser and the first to have multiple side-by-side linked views to show conserved syntenic relationships at different scales (as shown in Figure 2.2). It encodes for conservation using both connection and color, limiting the number of colors to eight, and using a circular layout with connected curves to reduce the number of variations. Within MizBee, the structure of datasets with syntenic blocks is broken into three main layers of scale, where the highest level is the genome, containing a list of chromosomes, the next level is the chromosome, having a list of blocks, and the last level is the block containing a list of conserved features.

MizBee has a view for each one of these scales, starting with the genome view, that uses a circular plot where source chromosomes are shown on the outer ring, and target chromosomes in the inner ring, using an edge bundling technique to enhance trends. The chromosome view enlarges the information of the genome view, by using the same connection color with a rectilinear area showing secondary genomic features. The block view shows the features and their matches using oriented glyphs and connected ribbons, along with a histogram of their similarity values. This view supports flipping to get rid of criss-crossings and zooming to enlarge the block. MizBee allows users to interact with the views, which subsequently updates the content of lower level views.

Even though MizBee research does not include any attempt to declutter its genome view, it hints at the effectiveness of decluttering synteny diagrams. First, it uses the edge bundling technique to reduce visual clutter by bundling together connections from neighboring blocks, making the blocks more noticeable and hence the layout clearer. Second, the MizBee paper highlights a case study where a collaborator used the viewer to develop an algorithm to find conserved syntenic blocks within stickleback and pufferfish fish genomes [23, 49]. MizBee authors describe that the collaborator was disappointed initially by seeing how cluttered his data was when visualizing it using MizBee, as he was using simple methods such as scatter plots, where the noise made it difficult to see the relationships. The algorithm was then refined to show the most important conserved syntenic relations. MizBee was used here as a visual inspection method to detect clutter from syntenic blocks, which certainly highlights the need for doing this detangling in place in the visualization, accelerating the comprehension and refinement process of the data.

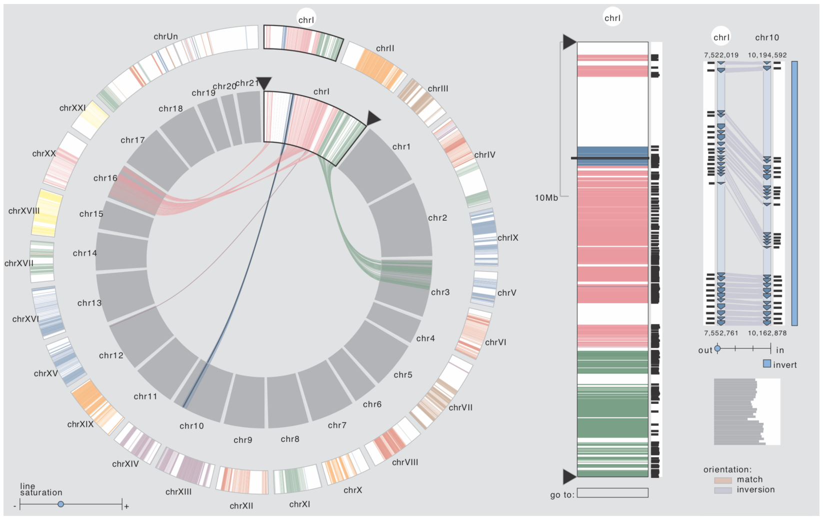


Figure 2.2: Example of the MizBee browser extracted from *Meyer et al.* [49].

mGSV (*multi-Genome Synteny Viewer*) [59] is a web-based synteny viewer created to visualize syntenic relationships between multiple genomes. Each genome is first displayed in a circular plot (similar to Circos [44]), showing the overview of all the conserved regions. From there, genomes are represented in different horizontal bars using two types of linear plots: "pairwise viewing mode," which shows only the conserved syntenic blocks between neighboring genomes, and "multiple viewing mode," which is the direct translation of the circular plot to a multiple linear plot by showing syntenic blocks among all bars (*genomes*).

mGSV uses a customized genome synteny data file as input, which specifies the conserved syntenic blocks locations for each genome. It also accepts an optional annotation file to show accompanying genomic features as an annotation track within the genome. Given that genomes are represented as linear bars, the viewer is limited to accept only one track per genome to avoid overcrowding the display. Although mGSV allows users to manually adjust the order of the genomes, it also includes a greedy heuristic algorithm to look for the optimal genome order based on the size of all conserved syntenic blocks between each genome pair, which improves visual clarity in both the pairwise and multiple viewing modes. However, being done on the server side of mGSV, this decluttering functionality can be time-consuming, hence it is not provided when big datasets are visualized.

SimpleSynteny [73] is a web-based synteny tool that visualizes syntenic relationships by using a horizontal linear layout having each genome in separate linear bars with gene locations inside. Homologous genes share a single color across genomes and are drawn as oriented glyphs connected by lines between all genome bars (similar to mGSV's multiple viewing mode). SimpleSynteny outputs the final layout as an image, taking two FASTA files [55] as inputs: a genome FASTA file, containing up to ten contigs (i.e. contiguous DNA sequences such as chromosomes), and a gene FASTA file containing up to sixty nucleotides or protein sequences. BLAST [2] is then used to align the gene sequences to the genome sequences, effectively getting syntenic blocks of similar genes.

SimpleSynteny uses a server to achieve calculations on the web, which makes processing larger genome files a limitation, needed to be split and processed separately. After the alignment is done, the configuration options allow users to change the look of the preview layout, being able to adjust the position and orientation of each contig for each genome by either moving or flipping them, respectively. However, each layout change requires a server request to generate a modified preview image, which is not ideal for achieving a pleasant interaction. SimpleSynteny also attempts to declutter synteny diagrams by reordering genomes, minimizing either the Euclidean distance between the gene connections or the number of inverted gene directions.

More recent examples of web-based synteny browsers include MultiSyn [3], Synteny Portal [45], and SynVisio [4]. These three browsers include configurable steps for the plots with basic interactivity, add support for one additional track or annotation file per genome, and allow users to save the final synteny plot as an image file. However, neither of them attempts to declutter their main synteny diagrams. MultiSyn and SynVisio use the results of MCScanX [77] to get the syntenic blocks, while Synteny Portal constructs syntenic blocks using prebuilt alignments from the UCSC genome browser database [26].

MultiSyn visualizes synteny by using a linear plot where each genome is represented as a linear bar connecting syntenic blocks from neighboring genomes (similar to mGSV's pairwise viewing mode). Both Synteny Portal and SynVisio are built using D3.js [5] and while they use an approach similar to MultiSyn for multi-level synteny analysis, they include other visualization options as well. Synteny Portal also uses Circos plots [44] where ribbons connect homologous blocks from genomes arranged around a circle. SynVisio also uses dot plots and introduces a novel way to visualize synteny using Hive plots [43], which are a type of network layout that arrange nodes (*syntenic blocks*) along radial axes. Moreover, SynVisio visualizes individual gene connections for each syntenic block using a block view with support for flipping and zooming (similar to MizBee). To conclude, SynVisio has a "snapshot" feature as well, to let users save their current view as a reference point to be able to revisit later.

2.3 Dataset

```
chr1  G0001.1  10    50
chr1  G0002.1  500   928
chr3  G0003.1  3440  3967
chr3  G0004.1  4598  5112
chr1  G0005.1  1441  2123
chr2  G0006.1  6810  7534
chr1  G0007.1  2981  3649
               .
               .
               .
```

(a) Genomic Feature Format (*GFF*) file.

```
chr1  1        400000   23
chr1  400001   800000   81
chr2  1        400000   17
chr2  400001   800000   94
chr2  800001   1200000  37
chr2  1200001  1600000  79
chr3  1        400000   55
               .
               .
               .
```

(b) Additional track (*BedGraph*) file.

```
## Alignment 0: score=398.0 e_value=4.9e-21 N=9 chrA1&chrA3 plus
  0-  0:  G262600.1   G404200.1        0
  0-  1:  G263100.1   G404400.1    2e-68
  0-  2:  G263300.1   G405000.1    4e-98
  ...
  0-  7:  G264700.1   G407400.1        0
  0-  8:  G265200.1   G408200.1   1e-165
```

(c) Collinearity (*MCScanX*) file.

Figure 2.3: Overview of sample generated files for the supported formats in AccuSyn.

This dataset section describes the three tab-delimited file formats that AccuSyn supports as input: GFF, collinearity, and BedGraph files. Figure 2.3 shows sample files for each format to see how the data is placed inside them. For the purpose of explaining the files, both the GFF and BedGraph samples shown in Figure 2.3 were created arbitrarily, while the collinearity sample shows part of the data for the first block of the Wheat genome [38].

Part (a) shows the main Genomic Feature Format (GFF) file which describes all the genomic features, giving the essential coordinate and length information. The simplified GFF uses four columns: chrID, featureID, featureStart, featureEnd, to describe the identifiers for both chromosomes and genomic features (*genes*), along with the start and end coordinates for each feature. Genomic features in the GFF file come in any order, so chromosomes and genes can overlap with one another, just as "chr1" overlaps with "chr3" in the example. However, AccuSyn visualizes syntenic genes by calculating each block and chromosome size using their minimum start and maximum end coordinates. To give the user more flexibility, AccuSyn also loads and extracts the same GFF information from a complete nine column GFF3 file.

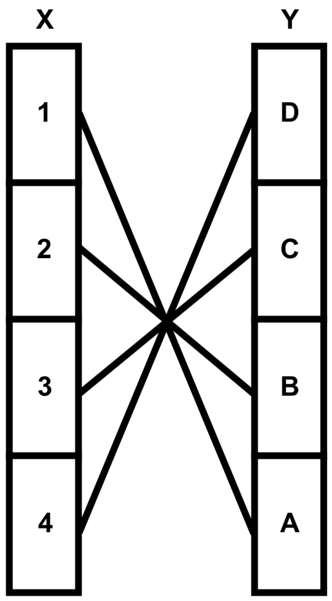
Part (b) presents the BedGraph file format used to load additional genomic data as tracks around the genome view. It uses four columns: chrID, chrStart, chrEnd, dataValue, to describe the chromosome identifier, window size, and data point to visualize. The window size is the arbitrary length interval chosen for the genomic regions on each additional track data value, using base pairs as the metric. For instance, Figure 2.3(b) is using a window size of 400Kb for each sampled value, which can be calculated for each row by using their two middle columns: $chrEnd - chrStart + 1$. Therefore, the window size needs to be consistent throughout each track. Each window encapsulates a bin of information. Additional tracks use these windows to visualize multiple values in frequencies, densities, or presence-absence analysis. Thus, biologists are able to gain insights for many genomic features when looking at these tracks, such as genes, transposable elements (TEs), copy-number variations (CNVs), i.e. number of copies of a particular genomic feature, and single-nucleotide polymorphisms (SNPs), i.e. DNA sequence variations that happen at a specific position in a genome.

Part (c) shows the MCScanX collinearity file, which is the second main file used along with the GFF file, to visualize syntenic relations. This file is divided into alignments (*blocks*) that contain detailed information for each syntenic link. The header line includes two calculated values: score and expectation value (*e-value*), describing the quality of the match and the statistical significance of the collinear block, respectively; and three informative values: number of connections, source and target chromosomes, and orientation of the block, using *minus* for inverted blocks and *plus* otherwise. The next lines define the individual pairs of similar gene connections for each block, including a numeric identifier for the connection, gene source and target identifiers, and the statistical significance of the connection (*e-value*). For the purpose of a synteny browser, the gene identifiers inside each block correspond to the position coordinates from the GFF file.

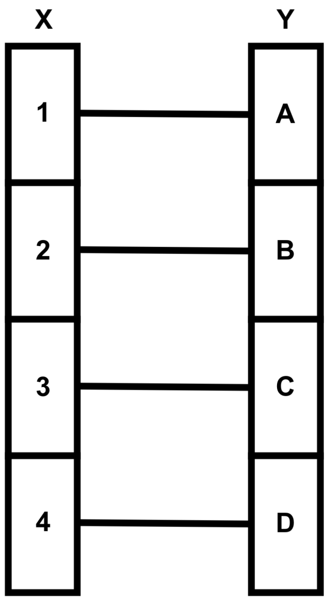
2.4 Crossing number

Now, we shift gears and change the theme of our background discussion to explain the crossing number problem, followed by a section on heuristic search. We are using Circos plots [44] to explore syntenic relations. The underlying structure that we are visualizing can be simplified as a syntenic graph where the chromosomes or genes are vertices and the conserved blocks are edges. In graph theory, the *crossing number*, $cr(G)$, of a graph G corresponds to the minimum number of edge crossings found among all graph embeddings of G, where a graph embedding is defined as a two-dimensional plane drawing where points are associated with vertices and curves (or line segments) with edges.

Testing whether a graph embedding has a crossing number $K = 0$, i.e. $cr(G) = 0$, is the same as testing whether a graph is planar, because a graph is planar if it can be drawn without edge crossings. Moreover, every planar graph has a straight-line drawing, that is, a graph embedding in which all edges are drawn as straight-line segments [11]. Hopcroft and Tarjan created an efficient algorithm to test for planarity in linear-time by the number of vertices in the graph [32]. Finding a graph embedding of G with at most K edge crossings, or in other words, answering the decision problem of "given a graph G and a non-negative integer K, is $cr(G) \leq K$?" is NP-complete (*nondeterministic polynomial time complete*) [15, 62]. This implies that there is no polynomial-time algorithm to solve the problem optimally (unless P = NP).



(a) Having four edge crossings.



(b) After minimizing edge crossings.

Figure 2.4: Simple bipartite graph example, where entities from set X are connected with entities from set Y.

The *bipartite crossing number*, $bcr(G)$, is a significant variation of the *crossing number* problem that looks for the minimum number of edge crossings in a bipartite graph G, that is, a graph whose vertices are divided into two independent sets, such that every edge connects a vertex from one set to the other. Essentially, looking for $bcr(G)$ is the problem of ordering the two sets of the bipartite graph G in such a way that the number of edge crossings is minimized. Computing $bcr(G)$ is also NP-complete, even when the ordering of the vertices of one set is fixed, which is known as the one-sided crossing minimization problem [62, 63].

Figure 2.4 shows an example of a bipartite graph made of two sets X and Y, each one with four vertices: 1, 2, 3, 4, and A, B, C, D, respectively. Part (a) shows the graph with four edge crossings when connecting the vertices from set X to Y. Part (b) shows another drawing of the same graph, this time having no edge crossings when inverting the order of the set Y. In our syntenic visualizations, we often see these bipartite plots where the sets X and Y correspond to two organisms (*genomes*), the vertices to their chromosomes, and the edges to their syntenic blocks that need to be minimized to remove clutter.

We are aware that there is a polynomial-time divide and conquer approximation algorithm for the *bipartite* *crossing number* problem within a factor of $O(log^2 n)$ [63] from the optimal graph, where n is the number of vertices (*chromosomes*). Nevertheless, given the fact that finding the *crossing number* problem, in general, is NP-complete, we decided to use our proposed heuristic solution (as defined later in this thesis) that can be adapted to any type of syntenic graph, not only bipartite ones.

2.5 Heuristic search

Decluttering automatically is a combinatorial challenge, which was the main challenge of this thesis. We wanted to take what we know about synteny and its cluttered visualizations, and apply a heuristic search algorithm to it. A heuristic is any method used as an approximation to find a reasonable solution to a problem. Old Artificial Intelligence references suggest that heuristics are the core of intelligence; indeed, one of the proposed names for the discipline of Artificial Intelligence was heuretics [60].

Heuristics are likely but not guaranteed to find a near-optimal solution. Therefore, they solve a problem quickly and are especially useful when algorithms that compute the optimal solution are too slow. A correct algorithm, for this case, might be a brute-force approach that would explore and consider all layout possibilities. However, such an approach is not feasible because we are dealing with a combinatorial problem, where our state space is every possible combination and each state corresponds to one layout. Thus, given our state of knowledge about the problem domain, we turn to heuristic methods for a practical solution.

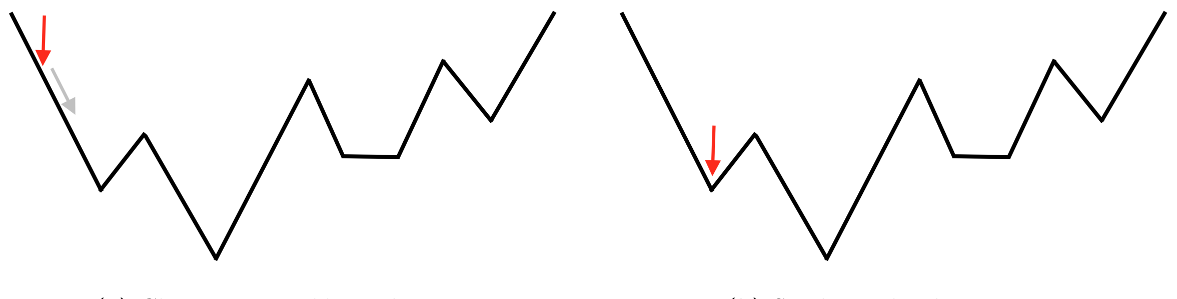
When minimizing syntenic crossings, the state space is implicit, because the set of all possible states is exponential in size, making it too large to generate and store in memory. Therefore, layouts are generated as they are explored and then discarded. For each state, dragging and flipping actions (formally described later in this work) generate subsequent states.

Heuristics are often designed for the solution of a particular problem, while more generalized heuristics, i.e. metaheuristics, are broader. The term metaheuristics was coined by Glover [19] to refer to techniques that are ranked above heuristics. More recently, Sörensen [68] defined a metaheuristic as “a high-level problem-independent algorithmic framework that provides a set of strategies to develop heuristic optimization algorithms.” Thus, by improving simpler heuristics, metaheuristics are able to find better solutions. For instance, if we establish a heuristic to find local optima (like Hill Climbing, described below), a metaheuristic built on top of Hill Climbing needs a way to overcome local optima. In this way, heuristics are “good guess” functions and metaheuristics are heuristics of heuristics, i.e. “good guess” frameworks that keep improving their heuristics or guesses over time.

We now briefly describe four popular methods: one heuristic, Hill Climbing, and three metaheuristics, Stochastic Hill Climbing, Simulated Annealing, and Tabu search. These metaheuristics can be classified as single-solution-based and stochastic by nature. The former specifies that its approach is to focus on iteratively modifying and improving a single solution at a time. The latter indicates that they are not guaranteed to consistently find an optimal solution even if one exists, i.e. their final solution is dependent on a set of random choices.

2.5.1 Hill Climbing

Hill Climbing, also known as *iterative improvement* [41] or *greedy local search* [61], is a classic blind search heuristic that starts at an arbitrary point and looks at "neighbors" (i.e. adjacent states) for better solutions, following a greedy heuristic by making a locally optimal choice at each step hoping to find the global optimum. If no better neighboring solutions are found, the algorithm stops and returns the best answer seen so far, which is likely to be a local optimum.



(a) Choosing a neighbor solution. **(b)** Stuck in a local optimum.

Figure 2.5: Example of Hill Climbing algorithm choosing a neighbor solution.

Figure 2.5 illustrates how Hill Climbing works by translating the solutions of a problem into a flipped mountain drawing. Since we are looking for minima, not maxima, we are "Valley Descending," rather than Hill Climbing. It is worth noting that depending on the authors, there are some slightly different definitions of Gradient Descent and Hill Climbing. We decided to stay with the latter because we are taking discrete steps, instead of using calculus to compute gradients. Explaining the algorithm this way makes it easier to understand the main purpose when decluttering a genome, by minimizing the number of link crossings. In reality, for this combinatorial problem, Hill Climbing would need to consider all neighbors for each state. When evaluating multiple neighboring solutions, this basic Hill Climbing deterministic heuristic always chooses the best-improving neighbor as the next state until no improvement is found at all. These examples were made considering only one neighbor at a time to simplify the idea of this local search heuristic.

By adapting the example to the problem of minimizing syntenic link crossings, one can see that the lower points all represent local optima (*minima*), while the lowest point of the mountain range is the global optimum, the best solution for the current set of chromosomes and block connections. Part (a) shows the hill climber starting at a random point marked as the red arrow. From there, it tries to look for better neighbor solutions, i.e. having fewer block collisions. The best choice here is to go down the hill as shown by the inclined gray arrow, choosing better solutions until arriving at the red arrow shown in part (b). The algorithm will finish on this local optimum spot because all neighbors represent worse solutions. For an algorithm to arrive at the global optimum, it would need to ascend first. In our problem of minimizing block crossings, similar to other optimization problems, the algorithm is not able to "see" the whole search space, just their current neighbors, which have more block collisions, therefore being worse solutions.

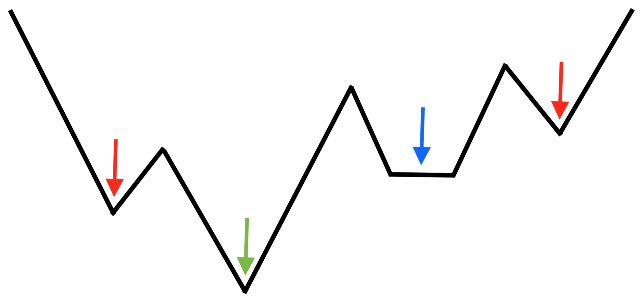


Figure 2.6: Example showing four local optima: two with red arrows, one with blue arrow (*plateau*), and one with green arrow (*global optimum*).

It is easy to see how this kind of approach works if there are no local minima, i.e. only one hill (*valley*). It will always find the minimum even with no knowledge of the terrain. Therefore, Hill Climbing is certain to get "stuck" on a particular area of the search space. Figure 2.6 has four local minima (red, blue, and green arrows) and only one global minimum (green arrow). The blue arrow represents a local minimum known as a "plateau." In this area, the same as with any local minimum, the algorithm is not able to determine the best direction to move, but with the difference that this is a flat search space where all the neighbors have the same value. Since there is no uphill or downhill to go, one way to go out of plateaux is to do a random search around until finding a neighbor with a different number of link crossings.

To sum up, given the complexity of our problem of minimizing syntenic link crossings, we believe that the probability of getting stuck in local optima is quite high when using the basic Hill Climbing heuristic. To go out of these optima, an ideal heuristic should occasionally accept poorer solutions, that is, go uphill momentarily, then make a new downhill move later.

2.5.2 Stochastic Hill Climbing

Stochastic Hill Climbing is a metaheuristic that extends the basic Hill Climbing by randomly selecting a neighbor for a candidate solution only if by chance an improvement is obtained [61]. Many authors have named this strategy differently: First-choice Hill Climbing [61], Random-mutation Hill Climbing [13], or simply Iterated Hill Climbing [50]. However, their main idea is the same, to add stochasticity to repeatedly choose the first better state from randomly generated neighbors.

One important variant is Random-restart Hill Climbing [61], also called Multi-start Hill Climbing [50]. It employs the adage: "If at first you do not succeed, try, try again." This variant involves two phases: first, it generates a series of random initial states and then it runs multiple Hill Climbing searches looking for the best result in all of them. Since there is no linkage between the searches, Random-restart Hill Climbing is considered a more general form of the basic stochastic Hill Climbing [50]. Russell and Norvig [61] consider

Random-restart Hill Climbing a complete algorithm with probability close to unity, because it eventually will arrive at the global optimum, but this comes with the downside of sometimes having to restart the algorithm many times. Nevertheless, as with Simulated Annealing (described below), Random-restart Hill Climbing can often find a good near-optimal solution after a small number of restarts.

2.5.3 Simulated Annealing

The annealing process in metallurgy involves heating and then cooling a metal very slowly to produce high-quality materials, improving its structural properties. The simulation of this process, in other problem domains, is known as *Simulated Annealing.* The main parameter is called temperature, meaning how high do we want to start the "heating process." There is also a ratio, that defines how fast we want to "cool down." Therefore, it has an analogy with an optimization procedure, where physical material states correspond to problem solutions, the energy of a state corresponds to the cost of a solution, and temperature and ratio control parameters.

In context, Simulated Annealing is a probabilistic metaheuristic technique used to find approximate global solutions to problems for which exact solutions are computationally difficult, i.e. NP-complete. It was independently discovered by Kirkpatrick et al. in 1983 [41] and by Černý in 1985 [6], using the direct relationship from annealing in metallurgy. Simulated Annealing can be used to solve instances of the traveling salesman problem: given a list of cities and the distances between them, finding the shortest route that visits each city and returns to the origin. It is a classic optimization problem that belongs to the NP-complete class, where there is no known polynomial-time solution. Both authors demonstrate how Simulated Annealing gets to near-optimal solutions for several instances of the traveling salesman problem, showing its potential as an optimization technique [6, 41].

The true origin of Simulated Annealing goes back to 1953 with the Metropolis algorithm, a simple algorithm that simulates thermostatistical equilibrium by showing the motion of atoms in contact with a heat bath at a temperature T [48]. To do this simulation the temperature is kept fixed, being a crucial difference when compared to Simulated Annealing, where there is an *annealing schedule* of declining temperatures.

Simulated Annealing is an enhanced version of stochastic Hill Climbing that combines the greedy heuristic with occasional random search, reducing the probability of becoming stuck in local optima, thus yielding to both efficiency and near completeness. The annealing process shows that just starting from a high temperature and then lowering it is not enough to find ground states of matter or the states with the lowest energy (i.e. minimal cost or optimum). Kirkpatrick et al. state that the best results are obtained when doing *careful* annealing, by first increasing the material's temperature to the melting point, then lowering the temperature slowly, and ultimately spending a long period of time in the vicinity of the freezing point [41]. If this is not done correctly, one can get stuck in locally optimal structures. The problem with Hill Climbing is that only accepting solutions that lower the cost function makes it behave like "quenching," the rapid cooling of a material, in the sense that achieving the best solution is usually unlikely. The metallurgy term quenching, therefore, has a direct analogy with local optima [39].

The same annealing analogy applies to Simulated Annealing, where the temperature regulates stochasticity, detrapping the search from poor local minima by allowing occasional *uphill* moves. The challenge though is to cool the temperature as quickly as possible (i.e. almost approaching a *quenching*), while preserving the guarantee that no local minima trapping occurs [72]. To illustrate this idea, Russell and Norvig [61] say that one can imagine the task of rolling a ball into a bumpy surface (like Figure 2.6 on page 17). If we let the ball roll, it eventually stops at a local minimum. The trick is to shake the surface hard enough to bounce the ball out of local minima but not so hard that it moves away from the global minimum. Simulated Annealing is all about "rolling the dice" by shaking hard at high temperatures and then gradually reduce the intensity of the shaking by taking fewer chances at lower temperatures. To go out of these local optima, Simulated Annealing employs the old saying: "*sometimes in order to keep moving forward you have to take a step back.*"

The search process of Simulated Annealing also has a close similarity with many real-life scenarios. For instance, one could also say that it is like filling an empty water bottle. At first, the water would be poured unconsciously, like when gathering all the information in a search space, and as the water reaches the top edge of the bottle, the pouring would be done carefully, like when choosing an optimal solution. The same happens to Simulated Annealing, starting by gathering all possible clues to get to know the surface of the problem (*search space*) and ending by putting all the pieces together with a constructive decision for the optimal solution.

2.5.4 Tabu search

Tabu search is an optimization metaheuristic that extends Hill Climbing and uses memory structures to take better decisions. It was originally developed by Glover between 1986 and 1989 [19, 20, 21]. Tabu means "forbidden" or "prohibited." The basic idea of Tabu search is to repeatedly construct a set of neighboring candidate solutions from the current state by using *tabu lists*, a short-term set of x previously visited solutions that cannot be revisited when exploring a neighborhood [61]. It applies *best improvement* to overcome local optimality by choosing the best available move at each iteration, i.e. always updating the current solution with the best neighbor from the set of candidate solutions, no matter if it represents a poorer solution. Tabu lists are then used to also prevent cycling when moving away from local optima, which helps in only looking at new solutions [9]. When finishing running a fixed number of iterations, Tabu search returns the best solution seen during the search process.

Tabu search can use stochasticity to generate a random permutation to improve the quality of the starting point or to randomly sample the neighborhood to a fixed size for each state when having a huge density of neighbors, like in our problem domain, minimizing syntenic crossings. However, given the magnitude of our problem, we speculate that the use of memory would be a difficulty for this domain and thus, Tabu search would not work significantly better. However, we do not rule out that a recasting of the problem may create good results.

2.6 Human-in-the-loop

We show the following examples to both describe and show the importance of having a human-in-the-loop approach for the process of detangling syntenic relations in AccuSyn. This approach is also called *mixed-initiative* [47], given that both a human and a computer system work together towards fulfilling a common goal.

One of the biggest examples of human-in-the-loop is Google's reCAPTCHA [74]. The acronym stands for "Completely Automated Public Turing test to tell Computers and Humans Apart." reCAPTCHA is a test used on websites where users are asked to prove they are human by interpreting two distorted words presented in an image, previously scanned from books, to block automated interactions such as spammers, cheaters, bots, or fraudulent behaviors. With reCAPTCHA, Google helped digitize old printed material by having users decipher scanned words that Optical Character Recognition (OCR) programs failed to recognize.

The main digitization purpose of reCAPTCHA, as explained by Luis von Ahn et al. [74], starts with getting a list of known and unknown scanned words, previously processed by computers using OCR programs. reCAPTCHA then takes one word from each list to show them as a test. Although each user has to type both words, they only need to identify the known word to be assumed as a "human." The answer to the unknown word is used to build Google's knowledge base for other words. When the unknown word is equally deciphered multiple times by many other users, it is then added to the list of known words for subsequent tests.

By continuously using this human-in-the-loop approach, the reCAPTCHA system achieved an accuracy of 99.1%, compared to 83.5% accuracy of standard OCR, when using a random sample from the New York Times archive and having 216 vs. 3,976 errors out of 24,080 tested words, respectively [74]. As of 2011, Google had already digitized the complete New York Times archive consisting of more than 13 million articles dating back to 1851 [24].

What Google essentially proves by using CAPTCHAS is that even though detection problems can be difficult for a computer to solve on its own, it seems easy and effortless for any user. Consequently, machines can improve their accuracy by leveraging and extending human intuition, knowledge, and intelligence, creating a powerful approach to solve any problem. Thus, Google continues to take advantage of CAPTCHAS by using human effort to go beyond text interpretation to image annotation to feed their machine learning models, which in consequence helps to solve hard artificial intelligence problems, like autonomous driving or multi-digit number recognition in Google Street View [22].

Within the field of artificial intelligence in machine learning, where computer programs look for patterns to automatically learn from data, a human-in-the-loop approach is constantly used to provide human input that can enhance the learning outcome. Andreas Holzinger refers to this as *interactive machine learning* (iML) [31]. This approach is useful in the health domain when solving computationally hard problems (e.g. NP-complete), such as protein folding, genome annotation, and for our research, genome decluttering, where human creative assistance and expertise help in reducing their exponential complexity. Therefore, even though AccuSyn is not directly using machine learning models to make decisions, by having a human-in-the-loop approach to this problem, we say that our tool is helping users “learn” syntenic patterns.

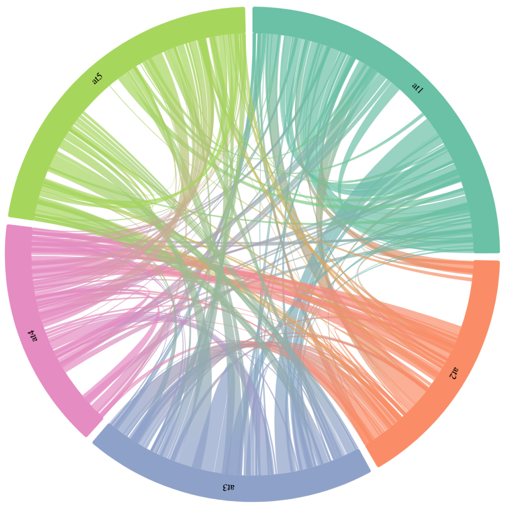
Another way to think about human-in-the-loop is as “cognitive prostheses” [12]. For example, some of us wear glasses or hearing aids, which do not fundamentally change us as humans, but amplify our vision and hearing senses. The idea is that artificial intelligence can be understood as something that enhances *our* cognition. From this perspective, AccuSyn’s approach towards decluttering syntenic relations is used as a way to enhance our synteny understanding, by providing an automated decluttering process, and to improve its outcome by having a human-centered approach.

3 Proposed Solution

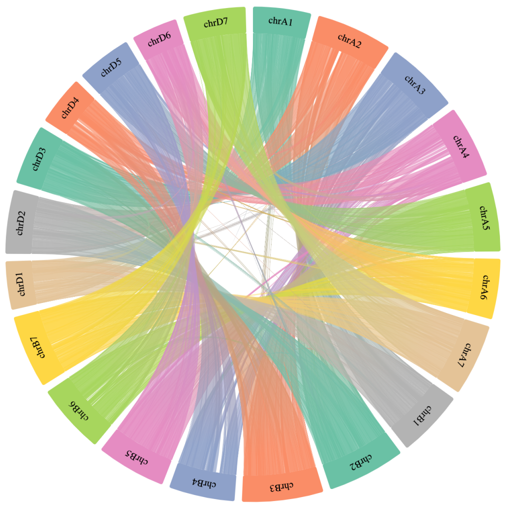
3.1 Decluttering operations

As even simple organisms have huge numbers of genomic features, syntenic plots present an enormous clutter of connections, making the structure difficult to understand. The first plant to be completely sequenced was *Arabidopsis thaliana* [37]. Figure 3.1(a) shows all the syntenic connections for this genome. One can see how simple the plant seems, by being a small genome with only five chromosomes, yet the visualization is cluttered producing 211 conserved blocks, each one connecting up to 188 pairs of genes.

On the other hand, Figure 3.1(b) shows the Chinese Spring Wheat genome (*Triticum aestivum*), a recently-sequenced plant genome [38]. It is clear to see how much more complex this genome is when compared to Arabidopsis. In fact, it has 21 chromosomes for a total of 1,299 syntenic blocks, each connecting up to 2,253 pairs of genes. However, it turns out the Wheat genome has three subgenomes A, B, and D, each with seven chromosomes, which allows even more chances for decluttering, as we will see later.



(a) Arabidopsis thaliana (Thale cress).



(b) Triticum aestivum (Wheat).

Figure 3.1: AccuSyn providing an overview of the syntenic links in *Arabidopsis* and *Chinese Spring Wheat* genomes.

To declutter a layout, a first step is to filter the connections using a threshold value, a user-defined parameter to show the blocks with “At Least” or “At Most” x connections, helping to visualize strong or weak syntenic relations. Filtering by block length helps to quickly focus on the main objective by eliminating unnecessary relations. It presents a clearer view of the possible patterns by providing a stripped-down version of a genome. All this is possible because the density of connections is based on the line thickness of the relationships. However, the filtered view has the downside of not showing the complete set of block connections. To alleviate this, users can rotate the circular view around its center point up to 360° by using a range slider, providing a natural way to look at the plot at any time. AccuSyn also includes two other operations: dragging and flipping a chromosome.

By default, AccuSyn displays chromosomes sequentially in alphabetical ascending order when visualizing a genome. If visualizing multiple genomes or subgenomes, chromosomes will be shown in the same order as well for each of them. AccuSyn allows the user to drag the chromosomes around the genome to reorganize the chromosomes in the most meaningful way.

The dragging operation works intuitively. The user clicks a chromosome and its connections are highlighted. From there, the clicked chromosome can be dragged anywhere around the genome while holding the left mouse button down. A clone of the dragged chromosome is created to get a pleasant interaction (as shown in Figure 3.2). In the end, the user just has to let go of the mouse button triggering a sequential animation that moves the chromosme positions one by one until the selected chromosome can slide over the new place.

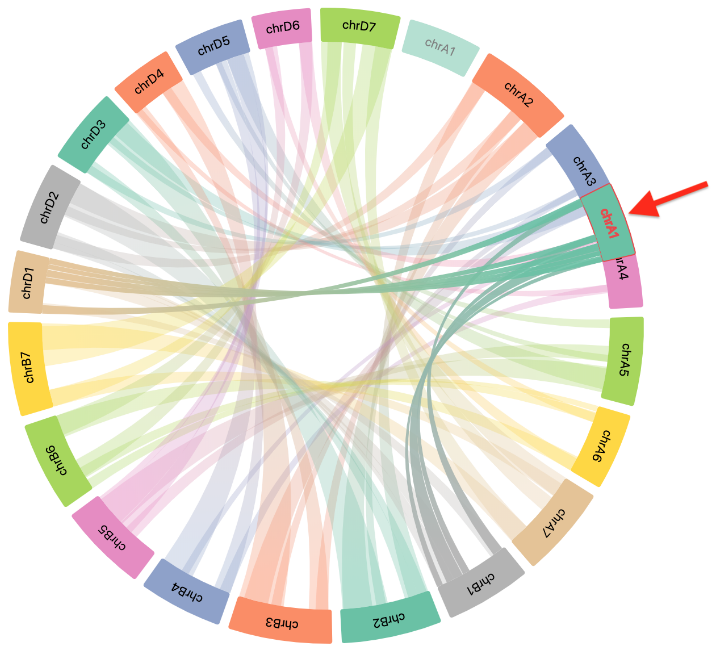
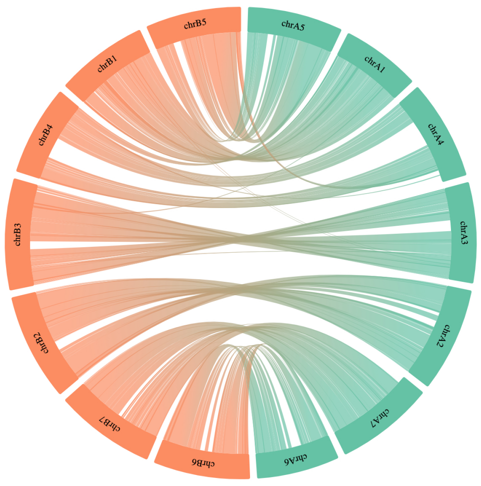


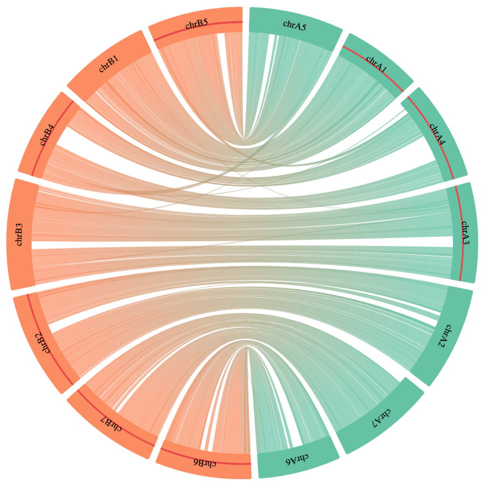
Figure 3.2: AccuSyn showing a dragging operation in the *Chinese Spring Wheat* genome.

Similar to dragging, flipping works by right-clicking on top of a chromosome. Flipping means inverting the locations of the blocks inside a chromosome, which helps reveal patterns. For instance, if a chromosome of length 100 normally has two blocks in the segments 0-20 and 50-85, then when doing flipping these blocks will translate to segments 80-100 and 15-50, respectively. Neither the content of the blocks or the order of the genes inside the blocks is altered when doing this operation. Figure 3.3 shows two pictures, comparing two Wheat subgenomes before and after flipping chromosomes A1, A3, A4, B2, B4, B5, B6, and B7. Flipped chromosomes are marked with an optional red line by default.

Our background chapter also presents an even simpler way to think of flipping with Figure 2.4 on page 14, where the entities 1, 2, 3, and 4 on the left side are connected to A, B, C, and D on the right side, respectively. Just by flipping the right side, we can see that flipping is all about reversing the original order, making it easier for the user to be able to visualize the relationships directly. We take advantage of the high probability of chromosomal inversions that organisms have throughout evolution [49], to also introduce flipping in the block view (as shown in Figure 4.5 on page 45). Therefore, when seeing these entities as two chromosomes X and Y, their connections can be thought as pairs of genes (as in the block view) or pairs of syntenic blocks (as in the genome view).



(a) Before flipping.



(b) After flipping.

Figure 3.3: AccuSyn providing an example of the flipping operation in the *Chinese Spring Wheat* subgenomes *A* and *B*.

The dragging and flipping operations decrease block collisions, link crossings or intersections showing in the plot, and thus, syntenic relations are shown more clearly. Figure 3.3 shows an improvement going from 3,444 block collisions on the left to just 1,105 on the right, due mostly to outliers or superimposed block collisions. The latter refers to those collisions that cannot be fixed because their origin or target positions are covering the same area (i.e. overlapping). Figure 3.4 shows the *Camelina sativa* genome [40], filtered to show the blocks with at least 940 connections in it, which leaves the genome with only 17 blocks but with 15 unfixable collisions because they are superimposed collisions, i.e. one on top of the other. Our goal is to minimize the number of non-superimposed block collisions.

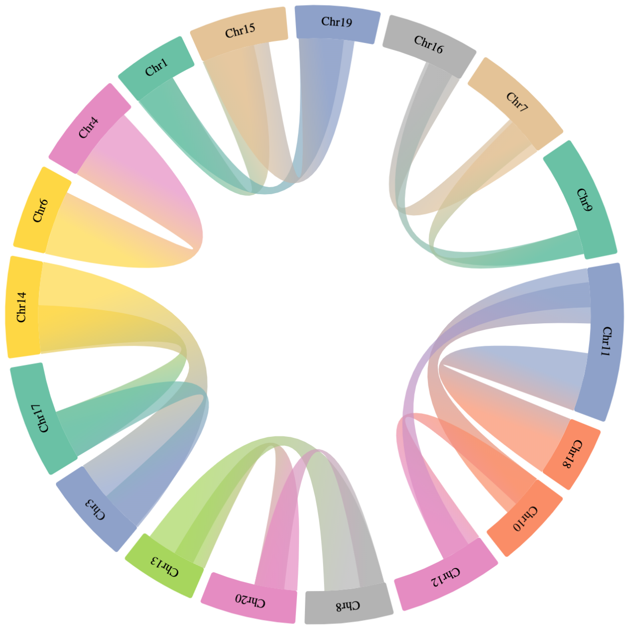


Figure 3.4: AccuSyn providing an example of superimposed block collisions in the *Camelina* genome.

3.2 Minimizing crossings

By using a circular plot (*Circos [44]*), chromosomes are represented as arcs and syntenic connections, represented as blocks, are drawn as curves. Although strictly speaking, a chord is a straight line segment joining two points on a circle, we will use the term chord here, rather than curve, to be consistent with the D3 documentation that refers to these diagrams as chord diagrams [5]. Chords make it harder to calculate the crossings exactly because the chord parameters are needed for each calculation. To simplify the calculation of the block collisions, AccuSyn projects each chord onto the line segments that connects their endpoints. Internally, each chord has a start and end angle for its source and target segments. We then use the cosine and sine functions to translate angles to (x, y) pairs. In the end, using the lines for all the possible combinations of source and target segments, we can easily determine how many block crossings are there in total in the plot by solving two linear equations of two variables for each case, a simple calculation.

By projecting each chord onto line segments when calculating block collisions, we assume that if two chords intersect, then their projection as lines would intersect as well. We introduce Figure 3.5 to explain

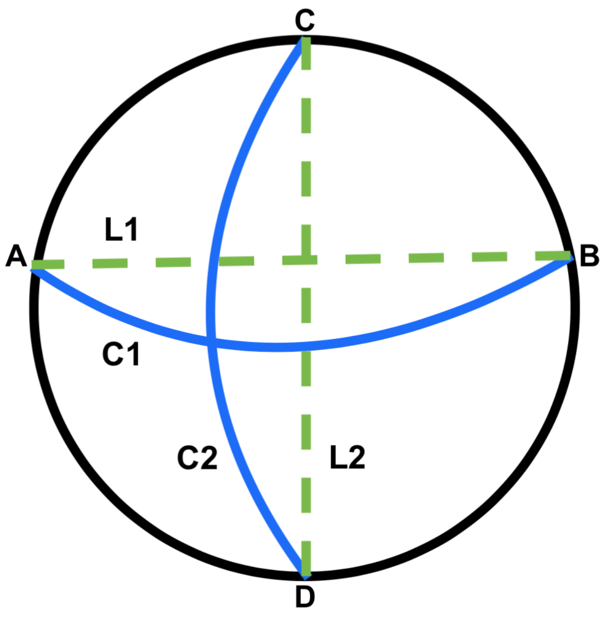


Figure 3.5: Example of chords, *C1* and *C2*, and their projection as line segments, *L1* and *L2*.

our idea using the described terminology of chords and lines. Let *C1* and *C2* be two chords inside a circle, and *L1* and *L2* be their corresponding line segments. Then, consider any projection P that maps *C1* to *L1*. Because *L1* divides the entire plane into two disjoint regions, each endpoint of *C2* is in one of these two disjoint regions. Therefore, any line segment connecting them must cross *L1*. Another simpler way to think about it is by looking at the endpoints of the chords. All chords connecting endpoints A-B and C-D have to cross each other, because of their positions in the circle. In practice, there might be some edge cases (*artifacts*) of the internal chord drawing algorithm from the D3 library that result on crossings not getting counted or being counted incorrectly, which makes this another good reason to have a human-in-the-loop approach (explained later in this chapter). However, this calculation has worked quite well for AccuSyn, making the calculation of block collisions much easier in general.

Dragging or flipping chromosomes helps in building a straightforward approach towards decluttering a genome. However, when doing these operations manually, users might take a lot of time to find an optimal set of drags and flips. Furthermore, getting an optimal solution to minimize collisions using a brute force approach would take years of time, as it is a combinatorial problem of trying all possible drags and flips and finding which configuration was optimal. Just trying all possible drags in Figure 3.1(b) on page 22 is an $n!$ operation that yields to $21! \approx 5^{19}$ possibilities, a huge number compared to today's genomic standards [69]. To avoid this cost, we considered an automated heuristic approach using Simulated Annealing.

3.2.1 Simulated Annealing algorithm

The Simulated Annealing heuristic finds a near-optimal solution to minimizing syntenic crossings in seconds. Many authors describe the Simulated Annealing procedure [9, 39, 61] as shown in Figure 3.6. The first step is to choose an initial solution, which is typically either the current configuration or an arbitrary layout of blocks and chromosomes. Then choose two initial parameters, temperature and ratio, to start the algorithm (step 3). These choices require some understanding of the problem domain.

```
1. Choose initial solution $S$.
2. Choose initial temperature $T$ and cooling ratio $r$.
3. While (T > 1.0) (not yet frozen):
        3.1. Pick a random neighbor $S'$ of $S$.
        3.2. Let $\Delta E = cost(S') - cost(S)$.
        3.3. If $\Delta E \leq 0$ (downhill move) then set $S = S'$.
        3.4. If $\Delta E > 0$ (uphill move) then set $S = S'$ with probability $P(\Delta E \mid T) = e^{-\Delta E / T}$.
        3.5. Set $T = r \times T$ (reduce temperature).
4. Return $S$.
```

Figure 3.6: Simulated Annealing computational procedure.

Simulated Annealing is composed of two stochastic procedures, as shown in steps 3.1 to 3.4, which are the core of the Metropolis algorithm [48], randomly perturbing the current state by generating and accepting new solutions. When generating new solutions (step 3.1), the algorithm looks for a neighboring solution by choosing between two tasks: swapping, where a neighbor solution is defined as swapping locations of two randomly chosen chromosomes, or flipping, where a neighbor solution is defined as randomly choosing one chromosome, and toggling its flipping state (i.e. if it was flipped, then unflip it, or vice versa).

The flipping task is considered only when the flipping frequency parameter is above 0%. Thus, our implementation of Simulated Annealing can either run the swapping and flipping operations simultaneously or separately, depending on the value f of the flipping frequency parameter: only swapping ($f = 0\%$), both operations ($0\% < f < 100\%$), or only flipping ($f = 100\%$). For example, having the flipping frequency at 40% means that 60% of the time the algorithm is going to do swapping and 40% of the time flipping.

In the algorithm, both the temperature (T) and ratio (r) can be translated into a fixed number of iterations. However, the ratio is used only as a cooling factor, to determine an annealing schedule or cooling scheme, i.e. how fast the algorithm finishes based on how high the starting temperature is. For example, if the temperature starts at 100 and the ratio is 0.9, the algorithm is going to run while the temperature is greater than 1, and after each iteration, the temperature is going to be reduced to 90% of the current temperature (using Equation 3.4).

Temperature plays an important role when deciding to accept a neighbor solution. When we talk about how good a solution is, we are talking about the energy of that solution, which comes from the annealing process in metallurgy, and is a domain-specific way of measuring the goodness of a solution. In our domain, where we are minimizing block crossings, energy is the number of block collisions for each possible solution (our objective function). Figure 3.6 (step 3.2) is about calculating the difference in energy (ΔE) between the neighbor and the current solution, i.e. the difference in block collisions between these two configurations (Equation 3.1). If a neighbor solution is better than the current solution, or in other words, if $\Delta E \leq 0$ (as shown in step 3.3), it represents a downhill move and it is accepted as current unconditionally.

On the other hand, if a neighbor solution represents an uphill move ($\Delta E > 0$) (step 3.4), then a uniformly distributed random number x between 0 and 1 is selected and compared with $P(\Delta E \,|\, T)$, the acceptance probability formula (as shown in Equation 3.2). This is the core of the second stochastic part of Simulated Annealing, how it decides when to choose a bad solution. If $P(\Delta E \,|\, T) > x$, the neighbor solution is used as the starting point for the next iteration with a new reduced temperature (step 4); if not, it is discarded, that is, not taken into account.

Table 3.1 and Figure 3.7 further show the idea of what happens when the algorithm is considering accepting an uphill move to a neighbor solution that is worse than the current one. An acceptance probability is calculated, where the acceptance rate is inversely proportional to the change in energy from one temperature state to another, i.e. the smaller the change in block collisions, and the higher the temperature, the more likely the algorithm will accept that solution. For example, if we have a temperature of 300,000 and an energy change of 100,000 block collisions between the current and the neighbor solution, then the algorithm has a 71.65% chance of accepting this neighbor solution as current. By having the same energy change, one can see that with a temperature of 20,000 there is only a 0.67% chance of accepting a neighbor solution. Essentially, if the temperature is high enough, the system will be more willing to accept poor solutions. Each time the algorithm accepts a poor solution, it expands its search space by considering other solutions outside of the current neighborhood of improving moves, which helps in decreasing the probability of getting stuck in local minima. As the temperature cools, only better solutions are accepted because the system is settling in an optimum and the chances of having a significant change in block collisions are smaller.

The probability formula for $P(\Delta E \,|\, T)$ in Equation 3.2 is the same acceptance probability used in statistical mechanics, the Boltzmann distribution [1, 64]. It is worth noting that according to the Metropolis procedure, if we continue accepting or discarding configurations and instead of reducing the temperature we kept it constant on Figure 3.6 (step 4), eventually the state would be in thermostatistical equilibrium, i.e. the probability of a change, $P(\Delta E \,|\, T)$, would tend to the Boltzmann distribution of the energy difference of two states [39]. Consequently, when using the probability formula in Equation 3.2, Simulated Annealing is sometimes called Boltzmann annealing [9].

$$\Delta E = \text{neighborEnergy} - \text{currentEnergy} \tag{3.1}$$

$$P(\Delta E \,|\, T) = e^{-\Delta E / T} \tag{3.2}$$

Being an approximation, the code has been written to allow a user to interact with the Simulated Annealing heuristic multiple times, giving the layout the chance to improve further. One can truly take advantage of Simulated Annealing when using high enough temperatures to give the algorithm enough time to explore the search space and settle in the optimum area. In the end, the algorithm will report the best solution seen so far, not necessarily the last solution that was seen.

Table 3.1: Sample values for the Simulated Annealing acceptance probability.

	Temperatures		
Changes	1,000	20,000	300,000
10	99.00%	99.95%	100.00%
50	95.12%	99.75%	99.98%
200	81.87%	99.00%	99.93%
2,000	13.53%	90.48%	99.34%
10,000	0.00%	60.65%	96.72%
20,000	0.00%	36.79%	93.55%
40,000	0.00%	13.53%	87.52%
60,000	0.00%	4.98%	81.87%
80,000	0.00%	1.83%	76.59%
100,000	0.00%	0.67%	71.65%

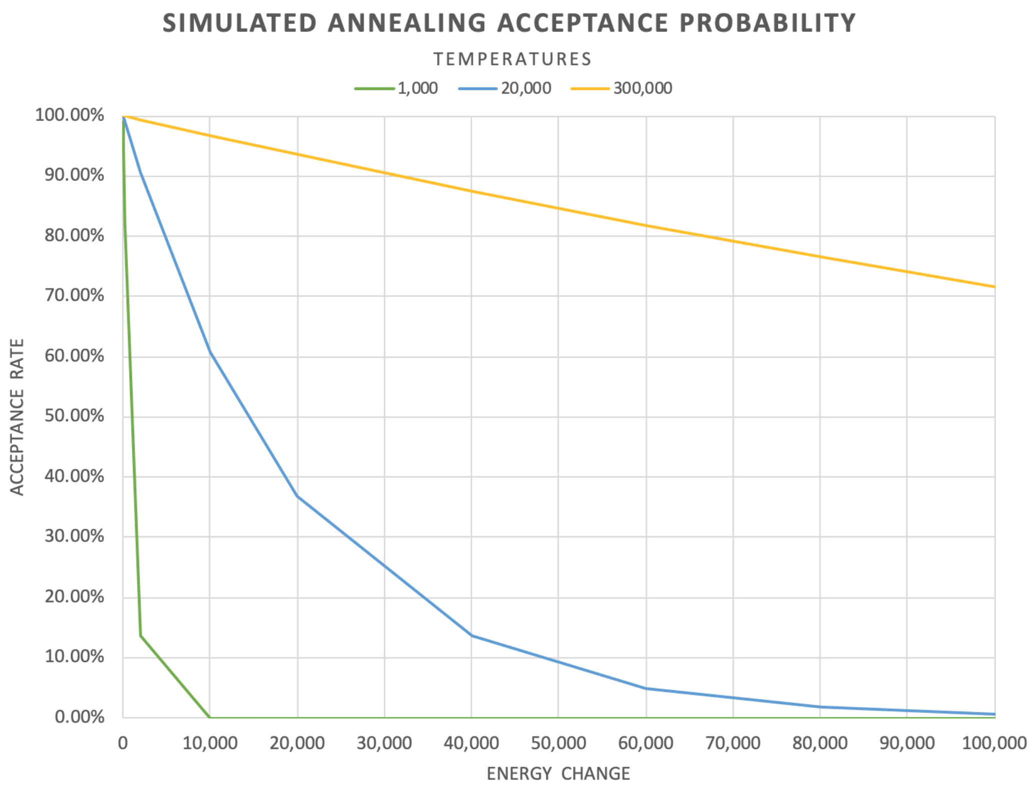


Figure 3.7: Simulated Annealing acceptance probability plot.

3.2.2 Annealing schedule

In its early days [41], Simulated Annealing was described as having two nested loops, the first one being the cooling loop where the temperature goes from high to low as shown in Figure 3.6 (step 3.5) on page 27, and the thermal equilibrium loop, where each configuration is tested using the current temperature until reaching thermostatistical equilibrium, i.e. the Metropolis procedure as shown in Figure 3.6 (steps 3.1 – 3.4) on page 27. The problem is how to simulate the idea of equilibrium. To address this, some authors have created an inner loop criterion based on the current temperature and neighborhood size [17], and others have used a fixed number of iterations or just run the loop until reaching a threshold level [41]. We tested the latter and did not encounter any significant improvements because the main idea of the algorithm of allowing some uphill moves to go out of local optima is still present. To follow the steps of some recent implementations [61] and to keep the procedure simple enough, given the magnitude of the problem, we use the Metropolis steps inside one cooling loop. Moreover, the core of the algorithm remains fundamentally unchanged.

$$T(t) = \frac{T_0}{ln(1+t)}, \quad t = 1, 2, 3, ... \tag{3.3}$$

As stated before, for Simulated Annealing to exit a local optimum, the temperature needs to be managed accordingly. The annealing cooling schedule is employed to specify a finite set of temperature values and is critical to the efficiency of the algorithm. If the temperature is reduced too quickly, convergence to a local minimum is more likely. If it is too slow, the algorithm will take a long time to converge [9]. Therefore, both the temperature and the cooling ratio play the role of control parameters. Geman and Geman [16] introduced the logarithmic cooling scheme shown in Equation 3.3, where T_0 is a large initial temperature. It was later proved that, if using this schedule, Simulated Annealing eventually converges to the global minimum with probability one [27].

$$T_t = \alpha\, T_{t-1}, \quad t = 1, 2, 3, ... \tag{3.4}$$

The problem is that the logarithmic annealing schedule is too slow. In practice, one must take into account three elements for the annealing schedule: the initial, decrement, and final value of the temperature [1, 30, 41]. The initial value T_0 is chosen high enough to give the algorithm the ability to get as close as possible to the global minimum. For the decrement, in practice, a fast exponential annealing schedule is applied as shown in Equation 3.4. The current temperature is defined in terms of the previous one and α is the *cooling ratio* constant that is usually close to 1. Common values for α are between 0.8 and 0.99 [9, 30]. The algorithm runs while the temperature is greater than 1 (Figure 3.6 (step 3) on page 27), which is when the algorithm, being a metaheuristic, produces near-optimal results for most problem instances. An exception can cause the algorithm to finish earlier, when the number of block crossings reaches 0 before the annealing schedule finishes.

Many other Simulated Annealing variants and annealing schedules have been proposed, studied, and compared [1, 9, 35, 36, 52, 72]. Particularly, Aarts and Korst [1] show an interesting proof where Simulated Annealing using the logarithmic cooling schedule needs an exponential computational complexity of $O(n^{n^{2n-1}})$ to guarantee convergence to an exact solution (*optimum*), compared to $O((n-1)!)$ for complete enumeration of all solutions. This proof shows that, when using the logarithmic cooling schedule, the number of samples needed for optimum convergence is larger than the size of the solution space. Therefore, we have chosen the exponential annealing schedule. Aside from it being more practical, it improves the convergence speed and it is in line with the main purpose of our thesis, automating the creation of synteny diagrams by providing an efficient and educated near-optimal guess.

3.2.3 Ideas from Stochastic Hill Climbing

Stochastic Hill Climbing is similar to Simulated Annealing, with the disadvantage that the user cannot control the search parameters, and thus, change the behavior of the algorithm, i.e. having a controlled random approach. From this point of view, Simulated Annealing incorporates two different behaviors. When starting the search with a high initial temperature, the algorithm behaves like *random search* by randomly choosing solutions and accepting most of them, even if they are bad solutions. After some time, as the temperature gets smaller and smaller, it changes the acceptance criteria of the algorithm and behaves more like stochastic Hill Climbing, by randomly choosing solutions only if they are better than the current one.

Due to its controlled stochastic approach, Simulated Annealing is a flexible algorithm that is sometimes considered an extension of stochastic Hill Climbing where some poorer moves are allowed [61]. When choosing the initial placement, the best that we can do in Simulated Annealing is either start with the current configuration or look for a random configuration. We decided to add the random-restart part of stochastic Hill Climbing by initially creating random solutions and choosing the best one as the initial placement. Table 3.2 shows the number of iterations that we are running based on the number of collisions for the initial layout, i.e. the more collisions, the fewer random-restart phases that we are contemplating. In the end, one could say that this sometimes is equivalent to running Simulated Annealing twice, because this process further improves our algorithm, by giving it a head start initially.

Table 3.2: Number of iterations used for the random-restart head start in Simulated Annealing.

Collision count	Iterations
0 – 100	500
101 – 500	200
501 – 10,000	100
10,001 – 50,000	50
50,001 – ∞	25

We took a second pragmatic decision to improve the behavior of the flipping task when looking for the best chromosomes to flip, i.e. the flipping frequency parameter is at 100%. Trying all possible flips for a particular synteny layout yields to 2^n possibilities, where n is the number of chromosomes, given that each chromosome can have two outcomes belonging to toggling its flipping state. Even though Simulated Annealing approximation does not need to look at every combination, it might not perform well when running only the flipping task because of two reasons. First, Simulated Annealing runs the same number of iterations based on the set parameters no matter the task that is chosen for a neighboring solution. Second, given that our implementation toggles the flipping state for one chromosome at a time, it can be hard for it to get out of bad solutions when having only the flipping task, i.e. we would need to consider a slower annealing cooling schedule (which is not practical as discussed above). All this means that, for this case, we might run Simulated Annealing for a long time and end up with no better layout found.

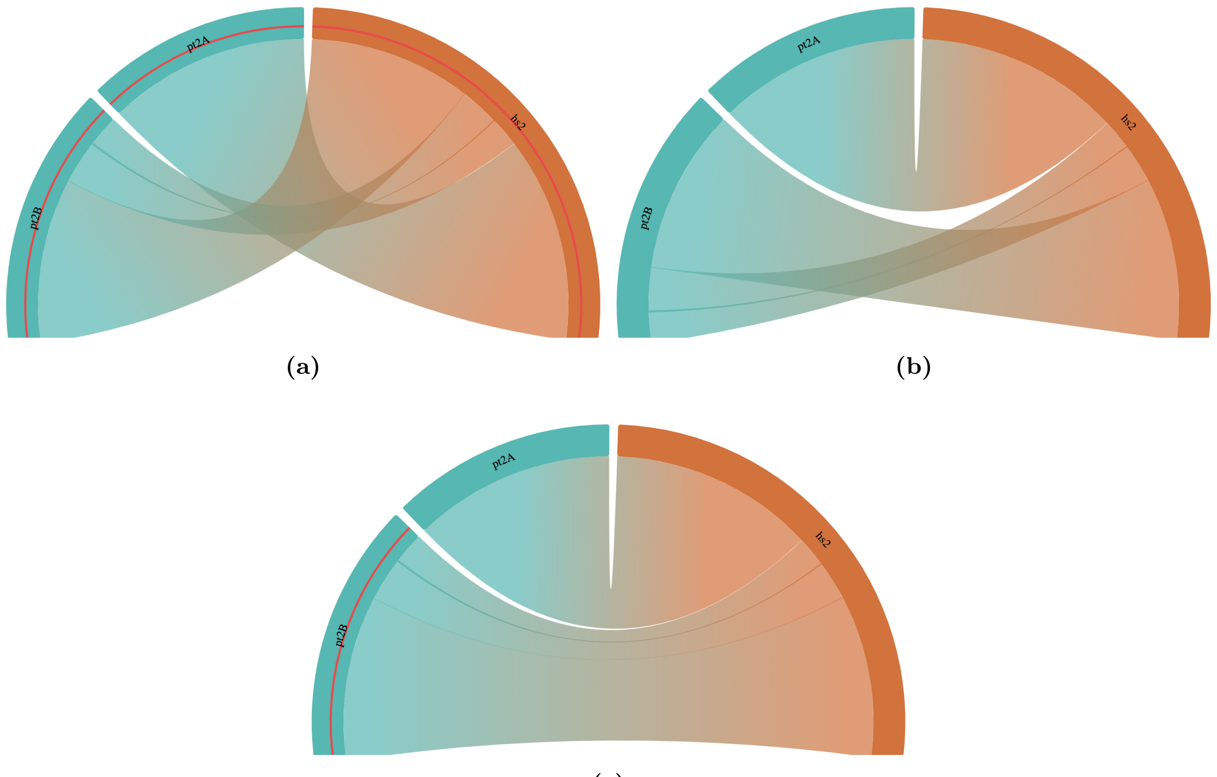


Figure 3.8: AccuSyn providing an example of the best flipping order for the chromosomes "pt2A," "pt2B," and "hs2" in the *Human* vs. *Chimpanzee* genome.

To alleviate this downside, we thought of a greedy strategy. Instead of running Simulated Annealing, we first flip all chromosomes in the current layout and then we check the collision count for each chromosome, unflipping the ones that make the final solution worse. This way, in theory, we would quickly end up with the best chromosomes to flip. However, given all the flip possibilities in any layout, we realized that the order in which we verify the chromosomes matters.

For instance, Figure 3.8 shows three chromosomes from the Human (*Homo sapiens (hs)*) vs. Chimpanzee (*Pan troglodytes (pt)*) genome. Because the chimpanzees have their second chromosome split in two, we are comparing "pt2A" and "pt2B" against the human chromosome "hs2." Part (a) shows these three chromosomes flipped. Part (b) shows the result of checking "pt2A," "pt2B," and "hs2," and part (c) "pt2A," "hs2," and "pt2B." When choosing the best order to verify whether a chromosome needs to be unflipped, we can see that chromosome "pt2A" has only one block, so its flipping state does not affect the final result, hence we can leave it unflipped. On the other hand, if we choose to check chromosome "pt2B" before "hs2," we end up unflipping both of them since that would give us the fewer number of collisions, even though it is not the best result (as seen in part (b)). Choosing to check "pt2B" after "hs2" is in fact the best result by leaving "pt2B" flipped (as seen in part (c)).

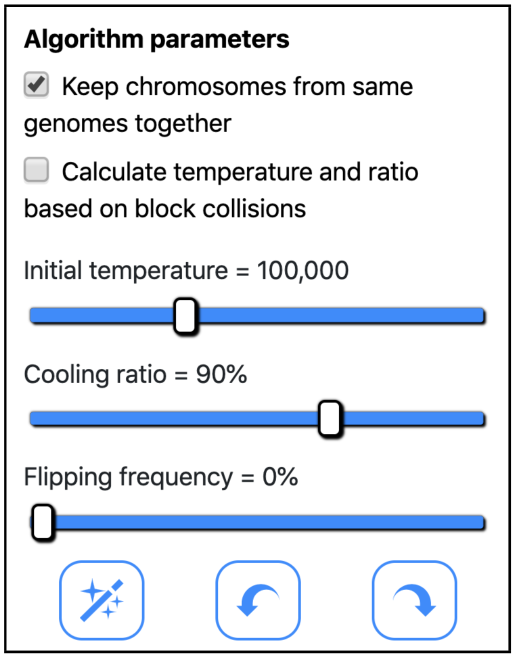
Consequently, we ended up modifying our greedy strategy by randomizing the order in which the chromosomes are going to be checked, instead of always checking the same order. Being partially stochastic, this approach does not always produce the best flips, but in practice just running it once shows a close approximation. Moreover, the verification process generally runs quickly since it does only n iterations, where n is the number of chromosomes. Nonetheless, this pragmatic decision is better compared to a complete stochastic process, i.e. flipping the conserved blocks of x randomly chosen chromosomes, given that we are assuming that all chromosomes are flipped at first and then randomizing only the verification process. The fact that we can run it multiple times, makes this approach similar to stochastic Hill Climbing or Simulated Annealing, being an effective solution to be able to quickly arrive at a near-optimal layout with the best flipped chromosomes.

3.3 Human-in-the-loop

To the best of our knowledge, no exact polynomial-time solution exists for every instance of our problem. However, our visualization allows the Simulated Annealing heuristic to be combined with manual dragging and flipping operations, as a "human-in-the-loop" approach. The user assists the algorithm with an educated decision, which might increase the chance of finding the global optimum. As users explore a genome and play with different settings, they will understand the syntenic structure more and more, being able to tune the algorithm parameters or use manual interventions. Hence, each time Simulated Annealing runs, users will get closer to meaningful insights along with an uncluttered layout.

Part of our human-in-the-loop contribution was to design an intuitive interface to the annealing algorithm and visualizing the output. Figure 3.9(a) shows our interface for the Simulated Annealing algorithm parameters. The first checkbox, "Keep chromosomes from same genomes together," appears only when visualizing more than one genome and allows users to restrict the behavior of the algorithm by not mixing positions belonging to different genomes when doing swapping operations.

The second checkbox, enabled by default, gives users the option to run the algorithm by precalculating the initial temperature and cooling ratio parameters. Table 3.3 on page 35 shows the default values chosen


Algorithm parameters

[✓] Keep chromosomes from same genomes together

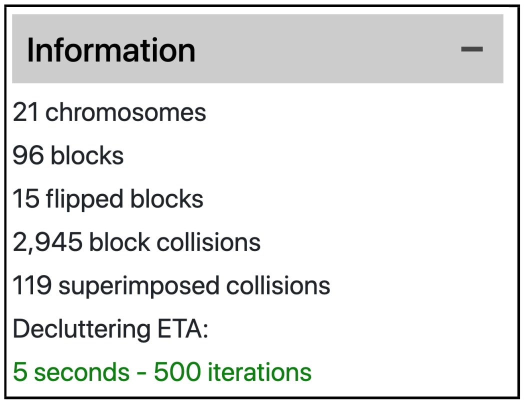
[ ] Calculate temperature and ratio based on block collisions

Initial temperature = 100,000

Cooling ratio = 90%

Flipping frequency = 0%

(a) Simulated Annealing parameters.


Information

21 chromosomes
96 blocks
15 flipped blocks
2,945 block collisions
119 superimposed collisions
Decluttering ETA:
5 seconds - 500 iterations

(b) Information panel.

Figure 3.9: AccuSyn interface for Simulated Annealing parameters and information panel.

based on the number of block collisions that the current layout has. Note that these defaults were created by varying the number of expected iterations from 100 to 4,000 iterations. The temperature always starts high enough, i.e. above 100,000, to give the algorithm enough time to better explore the search space. This option makes it easier for users not familiar with Simulated Annealing to run the algorithm without having to worry about parameters. However, although we tried to provide the best defaults, Simulated Annealing is sensitive to its temperature and ratio parameters and we are not aware of a way that the algorithm could automatically set the parameters for every dataset. The total number of iterations is also limited by how long it takes to calculate the number of non-superimposed block collisions in the current layout, i.e. the fewer collisions, the more iterations that we are contemplating by default.

Advanced users have the option of disabling the checkbox for default parameters, to adjust the initial parameters using range sliders depending on the instance of the problem. For the present work, AccuSyn gives users the choice to choose the cooling ratio between 0.7 and 0.99 and the initial temperature between 100 and 300,000. These values were chosen arbitrarily because of the exploratory nature of our research, giving advanced users the possibility of changing the total number of iterations as much as possible. The optional flipping parameter can be enabled by increasing the flipping frequency range slider above 0%, which automatically integrates the operation into Simulated Annealing for those layouts who might need it. We acknowledge that the interface could be simplified by using a range slider for the number of iterations and hiding the temperature and ratio annealing parameters from the user altogether, at the cost of denying users who might be familiar with Simulated Annealing the opportunity to test the parameters directly, and this is something that, if needed, could be tested empirically.

Finding a good outcome in an exploratory tool is a tough process. AccuSyn makes it easy to find meaningful relationships or to get publication-ready plots, by providing three buttons at the bottom of Figure 3.9(a). The first "magic" button runs the algorithm by minimizing block collisions. By keeping the interaction history for each layout, the last two buttons provide the ability to undo and redo all the decluttering operations, such as manually dragging or flipping, or automatically minimizing block collisions. This feature facilitates the understanding of syntenic structures as time passes, while further integrating human intuition by giving users the ability to go back and forth between enhancements.

Table 3.3: AccuSyn default temperature and cooling ratio values used for Simulated Annealing.

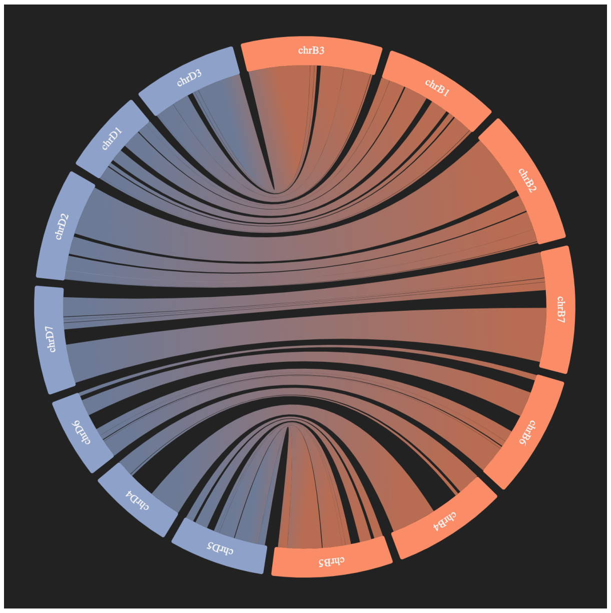
Collision count	Initial temperature	Cooling ratio	Iterations
0 – 50	165,500	99.7%	4,000
51 – 100	166,500	99.6%	3,000
101 – 500	170,000	99.2%	1,500
501 – 1,000	174,000	98.8%	1,000
1,001 – 10,000	111,000	97.7%	500
10,001 – 50,000	119,000	92.5%	150
50,001 – ∞	105,000	89%	100

In order to have a human-centered approach, the human has to know the specifics of the process and the decluttering progress. We wanted the user to be aware of everything, so we built an information panel (Figure 3.9(b)), which shows the number of chromosomes, blocks, flipped blocks, block collisions, and superimposed collisions that the current layout has. Finally, it shows an Estimated Time of Arrival (ETA) at the optimized layout.

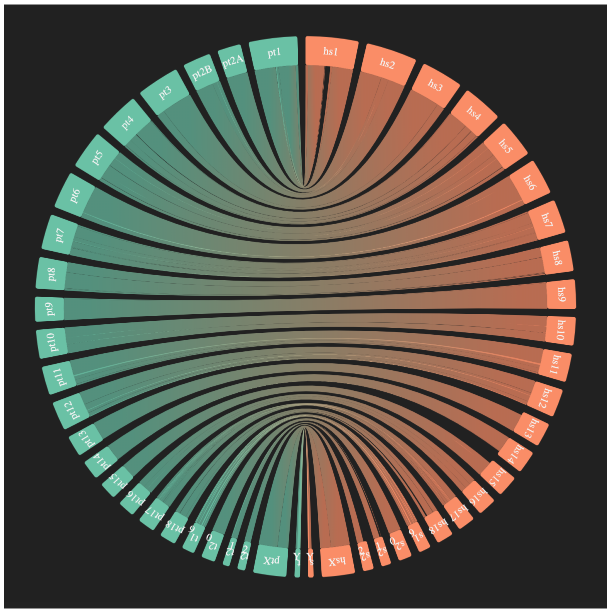
The decluttering ETA has a dual purpose inside the information panel (Figure 3.9(b)). First, it shows an approximation of how much time Simulated Annealing is going to take to declutter the current layout, based on how much time one iteration takes. Second, it shows the number of total iterations that the algorithm is going to perform, based on the initial temperature and cooling ratio values. The ETA label has three colors: green when the decluttering is going to take less than one minute, yellow between one and two minutes, and red for more than two minutes. An exception of the normal behavior of the ETA occurs when having the flipping frequency parameter at 100%, where the decluttering time gets smaller since the total number of iterations equals the number of chromosomes (as described above). With all this help, AccuSyn can quickly help users develop the best possible syntenic layout for any dataset.

In practice, what we have seen after running Simulated Annealing with multiple genomes is that weaker syntenic relations tend to follow the pattern of stronger relations because they are the most meaningful ones. Therefore, we have followed a trend when decluttering genomes. We filter the genome by block length, giving a manageable number of connections, and then we run Simulated Annealing multiple times, which is faster with fewer blocks. Finally, we undo the filtering and finish with the complete set of block connections, which should now be much cleaner. When testing with our current datasets, we found that running the algorithm two to three times was enough to get the best-decluttered stripped-down version of any genome, at which point, a new algorithm run did not provide substantial improvements unless there were some human interventions. This trend is shown in Figure 1.2 on page 3 in the introduction chapter of this thesis. AccuSyn provides an interactive way to combine human intuition and machine power, which makes human-in-the-loop useful because the same as we have discovered this trend, users can apply an intuitive approach towards decluttering any genome.

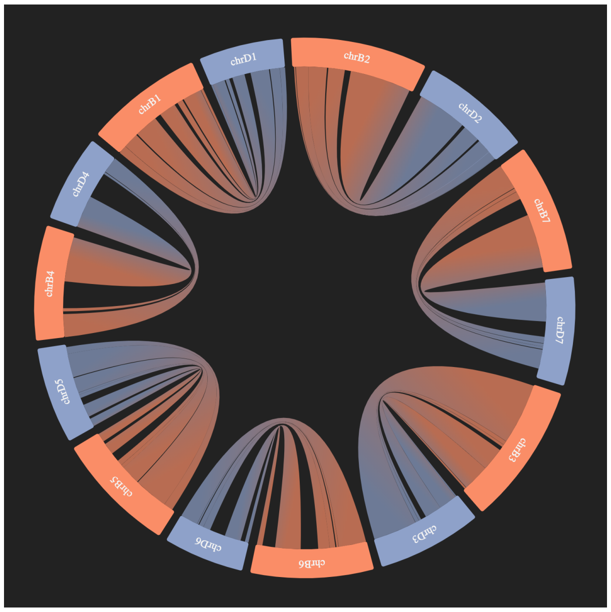
Using Simulated Annealing combined with human interventions, offers endless possibilities. When block crossings can be perfectly optimized, the genome view shows two kinds of clear streams: “basketball” or “bubble” views, as shown in Figure 3.10. Parts (a) and (b) show examples of “basketball” views, which in our experience happen in genome structures where there is a near-complete duplication of genes, thus giving strong syntenic relations for two genomes. For example, part (b) shows how similar human and chimpanzee genomes are. To obtain this result, the algorithm must be constrained to only swap positions belonging to the same genome. Parts (c) and (d) show “bubble” views. This is the default stream one gets after running Simulated Annealing without swapping restrictions, when there are multiple copies of the same genome structure (e.g. duplication or triplication). Moreover, Figure 5.1 on page 47 also shows “bubble” views, with triplicated and sextuplicated structures from the syntenic links of Camelina and Wheat genomes, respectively. These “basketball” and “bubble” views might help in interpreting many evolutionary events such as duplication and speciation, and also in finding evidence of a polyploidy state in these organisms.



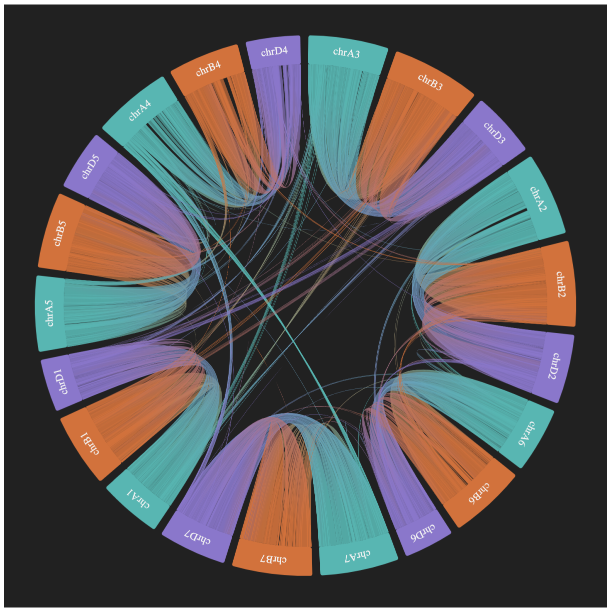
(a) Basketball view when grouping together chromosomes from *Wheat* subgenomes *B* and *D*.



(b) Basketball view when grouping together chromosomes from *Human* vs. *Chimpanzee* genomes.



(c) Bubble view when alternating chromosomes from *Wheat* subgenomes *B* and *D*.



(d) Bubble view on the complete triplicated *Wheat* genome.

Figure 3.10: AccuSyn creating "basketball" and "bubble" views in *Human* vs. *Chimpanzee* and *Chinese Spring Wheat* genomes.

4 Additional Considerations

4.1 Implementation details

AccuSyn uses two main features to be a web-based synteny browser capable of managing a large volume of genetic data: it is a single-page and client-side-only application. The former means that it interacts with a user by dynamically modifying the content of one page, instead of loading different pages from a server. The latter refers to the way that AccuSyn works in the browser, by storing all the content locally on the client computer instead of requesting resources from a server. This keeps sensitive genomic data secure by not sharing it with another computer.

Applications that use a client-server architecture, or a centralized model, have the advantage of allocating both the data and functionalities in one place, to be able to provide them to multiple users. Synteny browsers that follow this model include MultiSyn [3], Synteny Portal [45], mGSV [59], or SimpleSynteny [73]. However, these applications make it difficult to visualize a large dataset pleasingly because they do not allow for real-time interactions, i.e. information goes to the server which processes new changes to pass back to the client, limiting the smooth genome browsing experience [51].

AccuSyn depends on a server computer only for hosting the application content. Once a user (*client*) enters the website, the server loads the application content into the client browser, so that all the functionality is local to each user. Additionally, when a dataset is initially parsed locally, AccuSyn precomputes conserved syntenic relations so that as soon as the user interacts with any checkbox, button, or slider, updates immediately appear on the screen, offering an even richer experience.

This has been possible given the modern technologies that AccuSyn uses. Being a web application, it is essentially based on the three common web languages: JavaScript (JS) [14], HTML5 [76], and CSS3 [75] to provide interactivity, structure, and styling to the site, respectively. However, besides native Javascript, AccuSyn uses three JavaScript libraries to create the genome and the block view along with the configuration panel: D3.js [5], CircosJS [18], and React.js [33]. D3.js (*Data-Driven Documents*) [5] provides powerful and dynamic visualizations by efficiently manipulating the Document Object Model (DOM) (i.e. object representation of any HTML document), hence making it easier to visualize data with smooth animations and allow for interactivity. It uses Scalable Vector Graphics (SVG) to create resolution-independent visualizations (further described in section 4.4).

CircosJS [18] is the D3.js version of the Circos software [44] and it is used in AccuSyn as the starting point to create Circos plots for the genome view. This basic functionality is extended with the decluttering operations as discussed in chapter 3. React.js [33] helps in creating dynamic user interfaces, by efficiently updating and controlling the DOM (similar to D3.js). We are using React.js to create some of the additional components that integrate with AccuSyn, such as saving stamps as shown in Figure 4.3 on page 43. Moreover, AccuSyn employs Bootstrap [34] and SASS [10] to extend the functionality of CSS3 [75], to manage the page layout and styling format.

The implementation of AccuSyn follows the seven tasks of the Visual Information Seeking Mantra, a complete framework for designing Information Visualization applications [65], which includes: overview, filter, details-on-demand, zoom, relate, history, and extract. Furthermore, the use of aesthetically pleasing Circos plots in AccuSyn overcomes the limitations of equivalent linear representations when visualizing several relationships, by reducing the visual confusion and complexity when looking at genomes or chromosomes as arcs in a single circle, instead of multiple stacked horizontal lines [44, 51].

AccuSyn helps users to start exploring any genome by displaying all possible one-to-many block connections when selecting one chromosome in the genome view, giving an overview of how many conserved block connections the selected chromosome has with all the others. When selecting multiple chromosomes, it displays all the many-to-many block connections between them, having the option to show the conserved links at the chromosome level, by offering a bigger panorama of the chromosomes of interest that cannot be seen when having the complete genome.

For coloring, AccuSyn follows the specifications of ColorBrewer [28], a tool designed for choosing appropriate color schemes. The genome view uses categorical or qualitative color schemes, having eight colors to reduce cognitive load by differentiating the chromosomes. Block connections are represented with either an extra solid color or a combined color that displays a gradient transition between source and target chromosome colors for each relationship. Moreover, both flipped blocks and chromosomes can use a dual color scheme where blue and red indicate flipped and unflipped, respectively. Additional tracks use a sequential or continuous color scheme given that represent data values arranged from low to high. An optional dark mode can also be activated for the views to adopt a dark appearance to make the important features stand out against the darker backgrounds.

AccuSyn has been created as part of the research being done by P^{2}IRC[1] at the University of Saskatchewan. It is an open source software released under the MIT license[2], making it free to use and explore. AccuSyn browser is hosted in a university domain [3] and all the project source code is available on GitHub [4]. Now, we describe some additional features of AccuSyn: block view, additional tracks, exporting views, saving stamps, and animations. We were able to add these features to make it as complete and interactive as we could and to meet the interest and requirements of the P^{2}IRC[1] project.

[1] Plant Phenotyping and Imaging Research Centre: `https://p2irc.usask.ca/`.
[2] Massachusetts Institute of Technology (MIT) free software license: `https://opensource.org/licenses/MIT`.
[3] AccuSyn Website: `https://accusyn.usask.ca`.
[4] AccuSyn GitHub: `https://github.com/jorgenunezsiri/accusyn`.

4.2 Block view

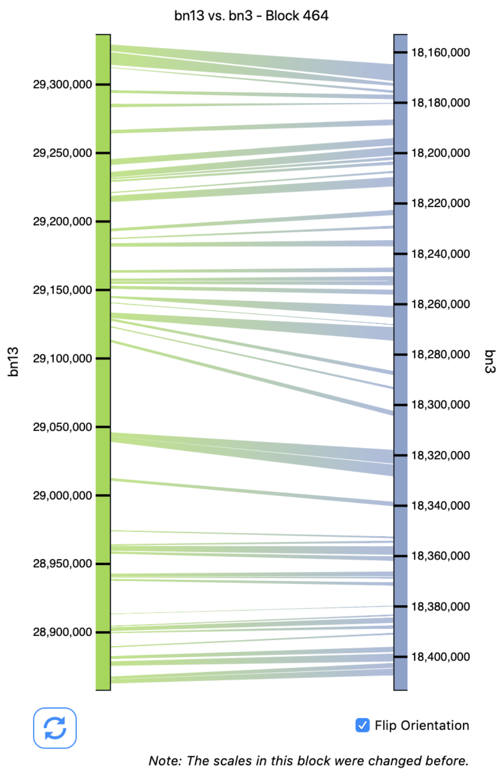


Figure 4.1: AccuSyn showing an example of a block view in the *Brassica napus* genome.

Hovering over a block in the genome view generates a block view on the right side of the screen showing all pairs of conserved gene locations, by using separate scales for each chromosome. This action also shows relevant connection information: identifier, score, e-value, size, and whether the block is flipped or not. Hovering over a gene relationship inside the block view shows the e-value and identifiers for both the connection and individual genes. Figure 4.1 shows a sample inverted block from the *Brassica napus* genome. The block view uses the same genome view coloring, where conserved gene relations are encoded as polygons connecting two parallel axes as a bipartite bar plot, representing chromosome portions with the exact coordinate information.

The block view effectively removes clutter and enhances conservation analysis by getting rid of criss-crossings when selecting the "Flip Orientation" checkbox in the bottom right corner. AccuSyn supports the ability to zoom in and out of the block view, by letting users drag up and down to better visualize the details. When a user zooms, the scales change accordingly enlarging the gene locations and providing a precise analysis by looking at specific conserved gene relationships. The scales can also be dragged independently, to align the connections in the block as necessary. Users are able to return to the default scale for any block by using the reset button in the bottom left corner. It is worth noting that AccuSyn also hints as soon as a block is not presenting its original scales, because each change to the block view is saved automatically, to make the comparison process easier and more efficient when looking at many blocks.

4.3 Additional tracks

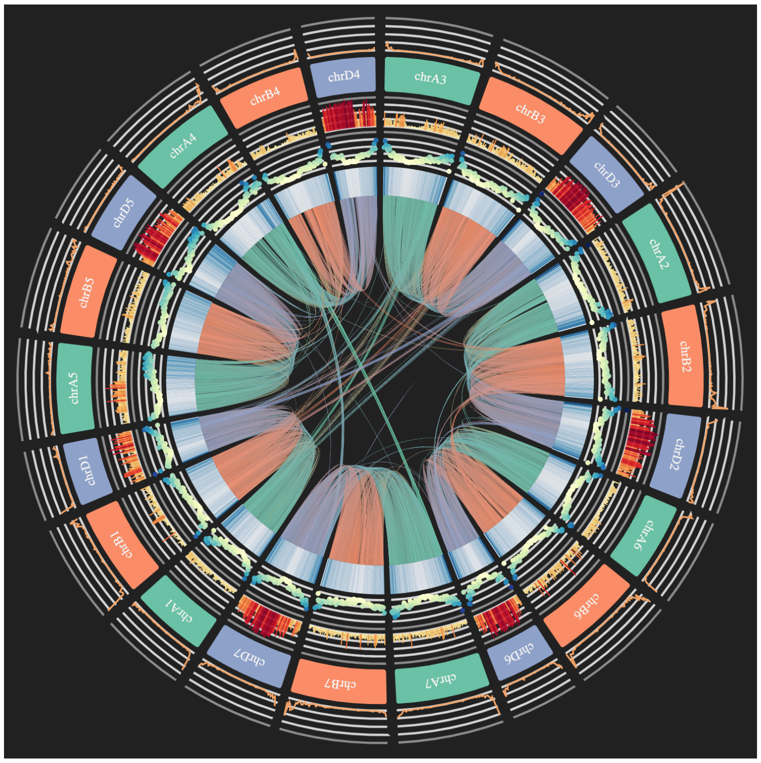
Accompanying genomic features are added into synteny diagrams as annotation tracks or additional tracks, present in other synteny tools such as Circos [44] and MizBee [49]. Part of the challenge for AccuSyn was to incorporate these tracks to interact with all the genomic data at once (i.e. syntenic links, block view, and additional tracks), improving the analysis of conserved synteny by helping to recognize patterns quickly. Being circular, multiple tracks easily integrate without overcrowding the display, avoiding limitations of linear layout viewers such as SynVisio [4] and mGSV [59], that allow only one track per genome. Therefore, AccuSyn follows the lead of Circos [44] to include multiple additional tracks around the genome view.

AccuSyn provides the ability to upload multiple tracks using the BedGraph file format (described in section 2.3 of the background chapter). Each additional track customizes the interpretation of a syntenic plot since any corroborative data is viewed all in one place. AccuSyn allows four types of additional tracks: heatmap, histogram, line, and scatter plot. Each offers different options to encode the data values. Heatmap uses a continuous color scale to show smaller values with lighter colors and larger values with darker colors; histogram and scatter plot use the same color scale and horizontal axis positioning with bars and dots, respectively; while the line type puts together the values in a continuous projection using a solid color. AccuSyn uniquely creates five horizontal axes, by using their minimum, center, and maximum data values, as well as, the midpoint between the minimum and center, and the center and maximum values, respectively.

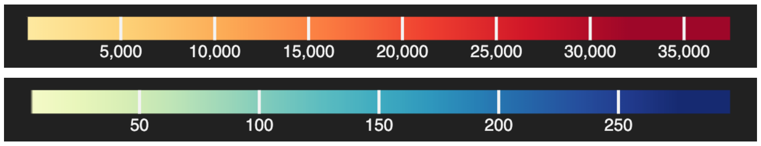
Figure 4.2 shows the Chinese Spring Wheat genome with four additional tracks. Part (a) highlights the layout which presents presence-absence of genes and SNPs relative to an artificial Wheat genome as line and histogram tracks, respectively, followed by gene frequency in both scatter plot and heatmap tracks. Except for the line type, each additional track supports a continuous color scale, whose values can be confirmed in a legend as shown in part (b) of Figure 4.2. Part (c) reveals the configuration panel which visualizes each track as a vertical tab, allowing the user to reorder, remove, or add new additional tracks. The content of each track shows dropdown options to change the track type, color scale, and placement, given that tracks can be positioned in the inner or outer portion of the genome view.

Being fully interactive, additional tracks loaded in AccuSyn respond to any interaction with the genome view. For instance, when chromosomes are dragged or flipped either manually or by the Simulated Annealing algorithm, the attached tracks also change their position and orientation, respectively. Hovering over a window shows the bin information, such as chromosome, value, and start and end coordinates. Users can also hover on top of the horizontal axes to show their calculated position value.

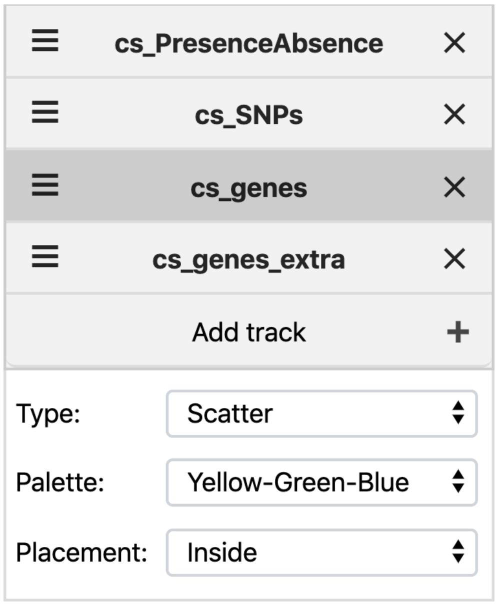
Even though users decide the best window size for the data on each additional track, AccuSyn provides a warning alert message when the loaded track uses a window size that produces more than 2,000 bins. In other words, if t is the total genome length in base pairs, the alert shows when any track has a window size smaller than $t/2,000$, effectively suggesting the ideal window size to use. This number was chosen arbitrarily, because having more than 2,000 bins on top of all the conserved relations, not only makes the rendering process slower, but also human eyes cannot appreciate that many details [65].



(a)



(b)



cs_PresenceAbsence

cs_SNPs

cs_genes

cs_genes_extra

Add track

Type: Scatter

Palette: Yellow-Green-Blue

Placement: Inside

(c)

Figure 4.2: AccuSyn showing an example of the *Chinese Spring Wheat* genome with multiple additional tracks.

4.4 Exporting views and saving stamps

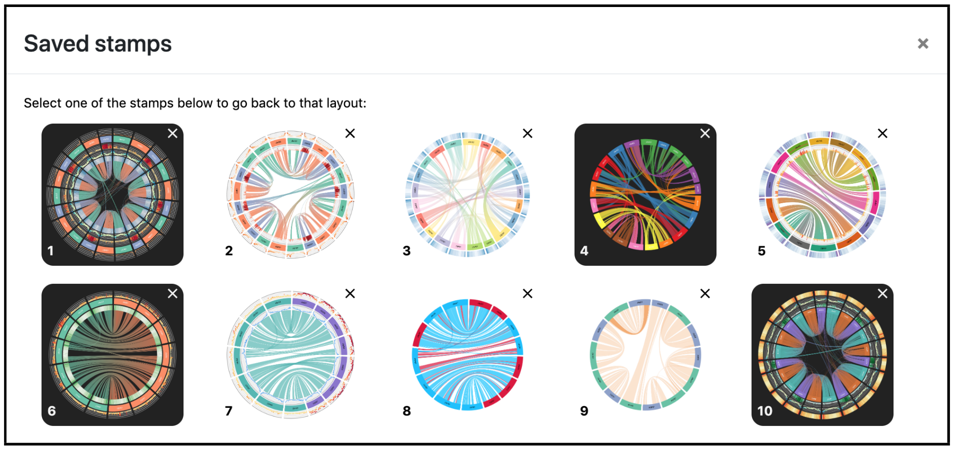


Figure 4.3: Saved stamps feature in AccuSyn.

Once the desired plot is obtained, AccuSyn allows the user to export the genome, legend, and block view (if present) at the click of a button by extracting the current plot configuration as an image using SVG, JPEG, or PNG file formats. The ability to output customizable plot images is included in other synteny browsers as well, e.g. MultiSyn [3], SynVisio [4], Circos [44], Synteny Portal [45], and SimpleSynteny [73]. The SVG format is often preferred as it can be scaled to different sizes in publication documents without losing quality. Furthermore, in AccuSyn, any changes made to the views are reflected when exporting them. For instance, if the dark mode is activated, then the images will download with a black background, or if a block gets selected in the genome view, then the user has an option to keep it highlighted.

AccuSyn includes an intuitive "save layout" functionality where the current configuration can be saved as stamps to be reloaded at another time. When being saved, these stamps essentially store *thumbnail* images, which are reduced-size static versions of the genome view. Therefore, saved stamps represent all the layout details just as they are found in their larger counterparts. This is an improvement compared to other implementations like SynVisio's "snapshot" feature [4], which uses numeric references for each saved view.

Figure 4.3 shows the saved stamps window with 10 different sample stamps shown in the order they were saved. Users select one of the stamps to go back to that layout, which reloads the genome view with the same characteristics as the chosen stamp. That is, stamps integrate any configuration option of the genome view, such as syntenic blocks color, genome palette, dark mode, highlighted block, number of filtered connections, degree of rotation, and added additional tracks.

Lastly, users can also delete any saved stamp when they do not need it. It is worth noting that if an additional track is included on any stamp, the track cannot be deleted unless the stamp is deleted first. This happens because saved stamps are meant to recover any available layout feature. Additionally, given that AccuSyn is a client-side-only application, the purpose of saved stamps is to allow for temporary saving plots within any session, while the exporting views functionality allows for permanently saving any view locally to a computer, especially when looking for publication-ready plots.

4.5 Animations

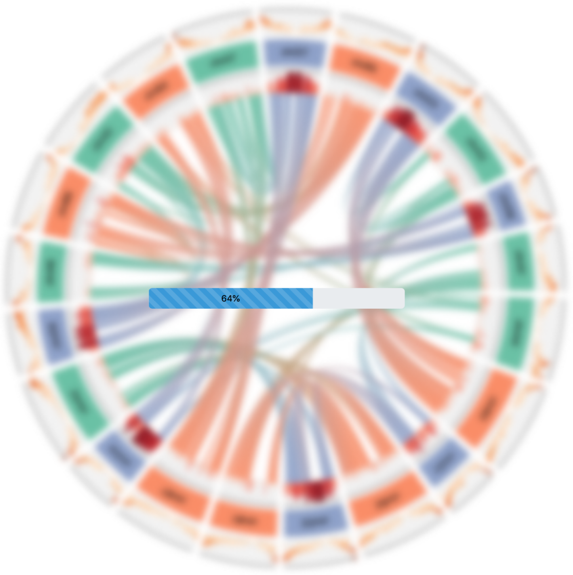
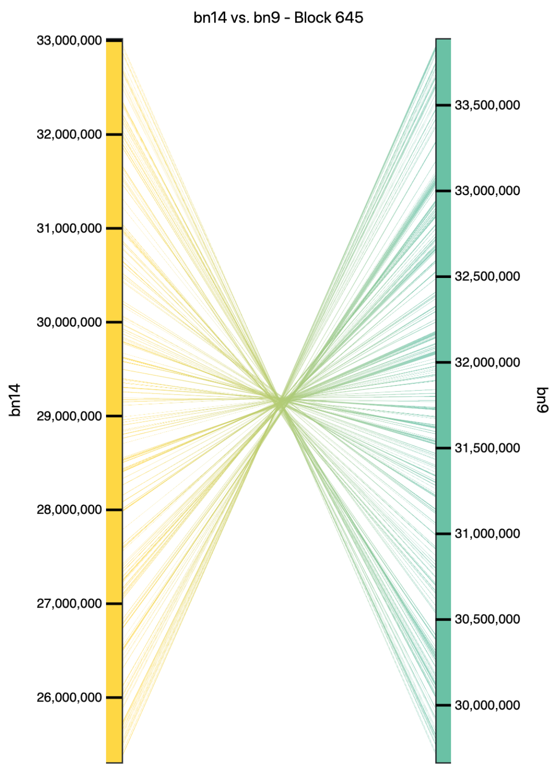


Figure 4.4: Progress bar when running Simulated Annealing.

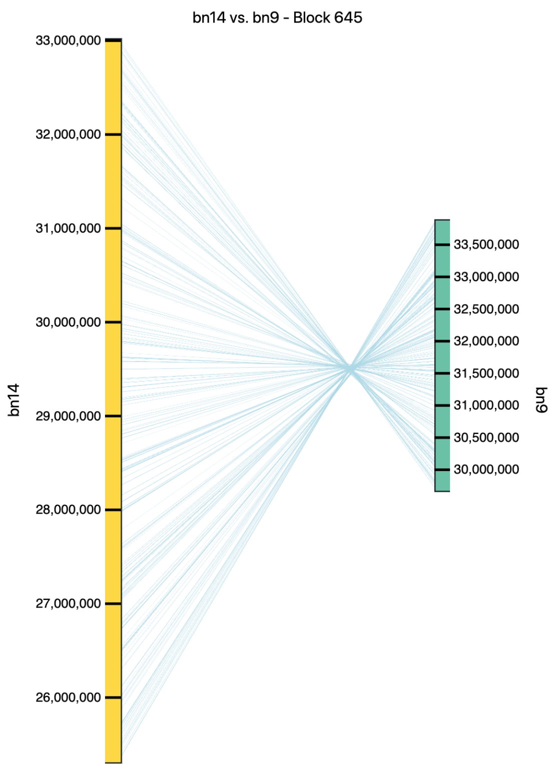
When the decluttering algorithm runs, Simulated Annealing receives the current layout and tries to produce an improved layout by minimizing collisions. This takes from a few seconds up to a few minutes, highlighted in the ETA label. However, most decluttering cases run quickly, making it impractical to show every combination that the algorithm tries. Therefore, AccuSyn shows a progress bar over a blurred view as the decluttering process runs, as shown in Figure 4.4.

The progress bar is based on the total number of iterations to declutter the current layout. Because of how D3.js [5] works, the animation is created by using the ETA time for one iteration to insert a timeout or delay into the loop. This gives the algorithm enough time to draw the gradual progress before proceeding to the next calculation. Otherwise, the first frame at 0% would jump to the last at 100% because of how quickly the browser processes the loop. When Simulated Annealing finishes, AccuSyn provides an animated transition from the initial layout to the best solution layout returned by the algorithm. This animation is a visual enhancement to give the user an understanding of how the syntenic layout is being rearranged. It is done by comparing to the initial layout in the most efficient way to first move all chromosomes that have an angle difference greater than zero and then flip all blocks with different coordinates.

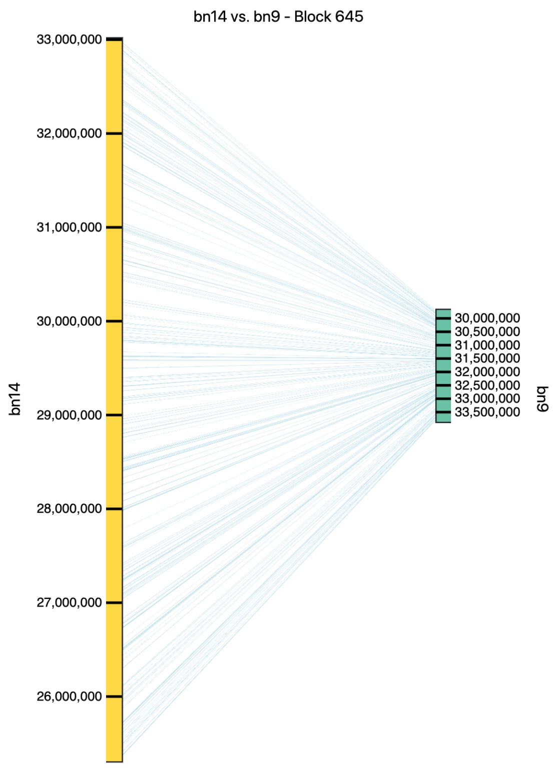
In a similar manner, animations are provided throughout AccuSyn, when a user wishes to move or flip a chromosome, or in the block view when getting rid of the criss-crossings by flipping the view. Figure 4.5 shows the steps on how the flipping animation works in the block view: first, the right axis shrinks until becoming very small; then, the axis flips by switching its endpoints; lastly, it expands to become normal again. This process is executed sequentially within a short period of time, so users can comprehend the flipping process. Overall, animations in AccuSyn provide essential visual cues to the observer, giving an accurate representation of the whole decluttering in both the genome and the block view.



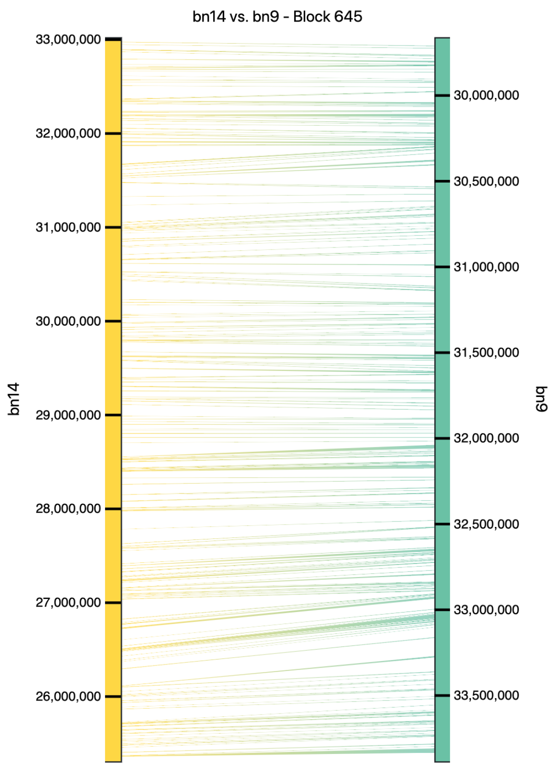
(a)



(b)



(c)



(d)

Figure 4.5: AccuSyn showing block view flipping animation in the *Brassica napus* genome.

5 Discussion

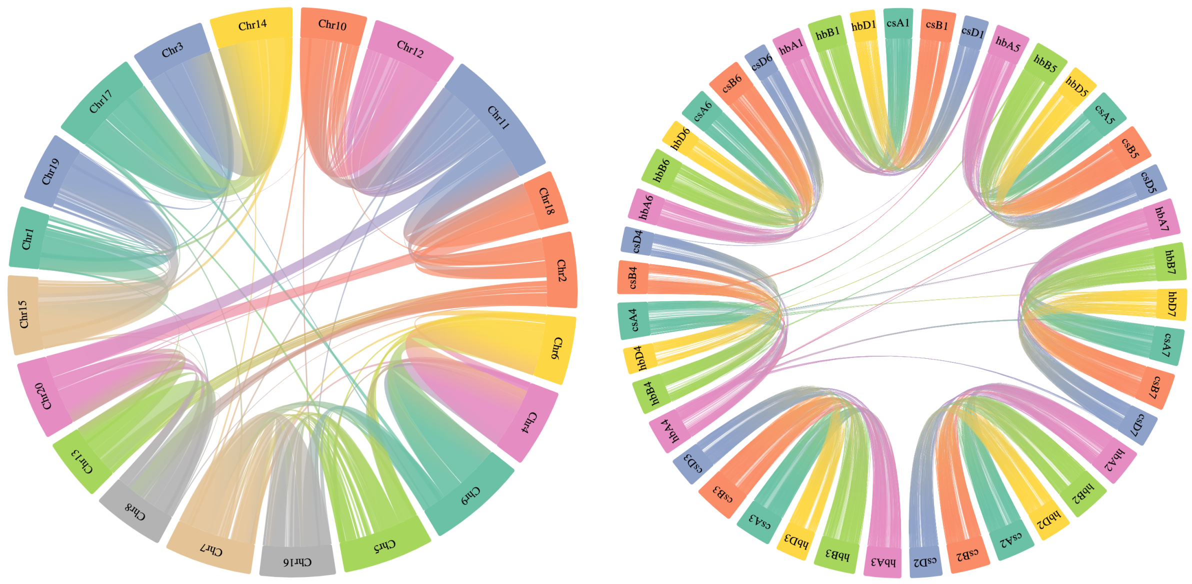
Initially, thinking about Simulated Annealing as a solution to decluttering a syntenic circular plot was hard because we needed to cast the decluttering problem onto the algorithm. Being a metaheuristic, Simulated Annealing is problem-independent and thus, has two requirements to make it work as an optimization procedure. First, we intuitively thought about ways to generate candidate solutions: swapping, to rearrange chromosomes in a meaningful way, and flipping, to consider the same chromosomal mutation that organisms experience throughout evolution. Second, we had to come up with a metric to measure the goodness of a solution (*energy*), which was the number of non-superimposed syntenic block collisions. Moreover, even though chord diagrams within D3 [5] have the option to sort the chords (*blocks*) within each chromosome to provide better ordering, one hard project constraint was to not move the block positions within chromosomes since they represent actual genome coordinates.

Simulated Annealing worked remarkably well by getting satisfactory visually appealing solutions, that appeared to be near-optimal, within a few seconds for most syntenic plots. However, this heuristic is not well-known in plant genetics, making it hard for biologists to take advantage of its power. We did our best to represent the ultimate best practices to build an interface in AccuSyn to be able to go back and forth between runs and to integrate the required algorithm parameters, such as the initial temperature, cooling ratio, and the type of operation: swapping, flipping, or both. This interface, with the help of the ETA information, was part of a human-in-the-loop approach that allows any user to change the parameters of the stochastic decluttering algorithm that produces the genome visualization.

The intuition behind Simulated Annealing is that initially the algorithm does not care if it is actually moving towards a good solution or not, and thus, it sometimes accepts bad configurations as well. Later, it becomes more careful and mostly selects good moves that reduce the number of syntenic crossings. From this perspective, Simulated Annealing has two limitations. It is not possible for the algorithm to finish earlier in case it runs x iterations without finding any better solutions. This is because of how the temperature regulates stochasticity, that is, the user might be terminating the algorithm as it just starts accepting better solutions. The only special case is when it arrives at the best possible layout, i.e. when the number of block collisions equals zero, which sometimes is not even achievable given there can be unfixable superimposed collisions.

Likewise, although Simulated Annealing easily goes from bad to reasonable solutions, it is much harder for it to go from a reasonable solution to the best one. An example of this can be a syntenic layout having only a few crossings fixable by applying either dragging or flipping operations to the affected chromosomes.

Given the nature of the algorithm, an overly fast annealing schedule might not generate enough iterations, preventing the algorithm from increasing its search space. A possible automatic way of overcoming this issue might be to run the algorithm many times. However, the fewer the crossings, the more obvious they are to users, making human interventions ideal for untangling certain configurations immediately.



(a) Camelina sativa genome. **(b)** Wheat - Chinese Spring vs. Artificial Hexaploid

Figure 5.1: AccuSyn providing an overview of the syntenic links in *Camelina* and *Wheat* genomes.

Nonetheless, decluttering is a useful feature for whole-genome comparative analysis, especially when relationships between chromosomes are unknown. Given an unknown structure, biologists usually do a BLAST [2] pair-wise comparison with the closest relative of that particular individual. This provides them insight into the relationships that they find with MCScanX [77]. However, sometimes a few chromosomes might not fit in the syntenic structure. For instance, Figure 5.1(a) shows how the *Camelina sativa* genome almost forms a complete triplicated structure, but by having 20 chromosomes in total, there are two chromosomes that do not follow completely this behavior: "Chr2" and "Chr18." AccuSyn helps any user see what is actually happening with chromosomes that do not fit as expected, while quickly confirming the genome structure that biologists were looking for.

Popular synteny tools like Circos [44] and Mizbee [49] provided the insight that circular plots produce a compelling visualization for looking at the similarities between syntenic blocks. However, there has been very little work on our main problem of detangling synteny diagrams. Both of these synteny browsers required a lot of tedious manual work to rearrange the genome, i.e. biologists had to know the structure in advance to be able to plot the optimal order of chromosomes. Moreover, the decluttering approaches proposed by mGSV [59] and SimpleSynteny [73] were done entirely on the server side, creating a time-consuming user interaction and hence, a limited functionality by being suitable only for small datasets.

On the other hand, given how powerful today's web technologies are, i.e. browser engines and front-end libraries, AccuSyn offers a better user interaction without a server. When visualizing multiple genomes, our tool helps in the decluttering process no matter the number of chromosomes. To confirm this, Figure 5.1(b) and Figure 3.10(b) on page 37 show the decluttered syntenic links of two Wheat genomes, Chinese Spring (*cs*) vs. Artificial Hexaploid Hybrid (*hb*), with 42 chromosomes, and Human vs. Chimpanzee genomes, with 49 chromosomes, respectively. It is worth noting that in general, when looking at Wheat genomes in Figure 5.1(b), the chromosomes get perfectly clustered by their subgenome number, i.e. all number one chromosomes get together and so forth. This might indicate that the disentanglement is deterministic but the Camelina genome in Figure 5.1(a) proves otherwise, making a brute-force approach not feasible.

5.1 User community

Throughout the development process of AccuSyn, we met with the biologists (*users*) on a routine basis (i.e. every two months) to discuss new requirements and share thoughts and feedback. For this reason, AccuSyn is already actively used in the research being done by P^{2}IRC[1] at the University of Saskatchewan, especially in research regarding the recently-sequenced Wheat genome [38].

AccuSyn was recently used to show 12 varieties of Wheat genomes at the 1st International Wheat Congress[2], including ArinaLrFor, Artificial Hexaploid Hybrid, Chinese Spring, Jagger, Julius, Lancer, CDC Landmark, CDC Stanley, Mace, Norin 61, Spelt, and SY Mattis. Moreover, a landing page was also created[3] to be able to pick any possible combination (i.e. source and target) of these Wheat genomes and load either AccuSyn or SynVisio [4] to visualize the conserved syntenic relations between them.

Overall, we have received positive feedback from everyone that uses AccuSyn for whole-genome comparative analysis. Apart from the regular internal meetings with the P^{2}IRC[1] group, AccuSyn has also been highlighted as one of the genome visualization tools being developed at the University of Saskatchewan at the CORS 2019[4], PAG 2019[5], and Phenome 2019[6] conferences.

5.2 Decluttering results

Table 5.1 gives an idea of how quickly the decluttering process can be done for any genome and allows future researchers to compare their results to AccuSyn. Note that these have been obtained informally rather than with naive users in an experimental setting. We are using five genomes as already presented throughout this thesis: Human (*hs*) vs. Chimpanzee (*pt*), Wheat subgenomes B and D, and the complete Wheat genome as

[1] Plant Phenotyping and Imaging Research Centre: `https://p2irc.usask.ca/`.
[2] 1st International Wheat Congress: `https://2019iwc.ca/`.
[3] 10+ Wheat Genomes Visualizer: `https://kiranbandi.github.io/10wheatgenomes/`.
[4] Canadian Operational Research Society (CORS) 2019 conference: `https://www.cors2019.ca/`.
[5] Plant and Animal Genome (PAG) 2019 conference: `https://www.intlpag.org/2019`.
[6] Phenome 2019 conference: `http://phenome2019.org/`.

shown in Figure 3.10 on page 37; Camelina sativa and two Wheat genomes, Chinese Spring (*cs*) vs. Artificial Hexaploid Hybrid (*hb*), as shown in Figure 5.1 on page 47.

Table 5.1: Table of decluttering results using AccuSyn for five different genomes.

Genomes	Initial number			Final averages			
	Chromosomes	Blocks	Crossings	Decluttering time (s)	Manual time (s)	Crossings	Improvement
Hs vs. Pt	49	135	8,952	48.72	33.79	244	97%
Camelina sativa	20	1,104	198,425	51.87	28.22	137,151	31%
Wheat (B, D)	14	418	65,825	7	4.34	9,933	85%
Wheat (A, B, D)	21	1,299	477,895	45.51	19.98	79,805	83%
Cs vs. Hb	42	6,915	12,695,156	151.50	108.67	1,445,820	89%

We did our test with a process very similar to Figure 1.2 on page 3 in the introduction of this thesis where the complete Wheat genome is presented with before and after images. We first filter the view to show the blocks that have at least 200 genes. Then, we use this filtered view to run Simulated Annealing with 25% flipping, and after that apply manual interventions. Finally, we undo the filtering to achieve the expected bubble or basketball view. Given that we are using a stochastic algorithm, this whole process is done three times for each genome shown in Table 5.1. Note that there is an exception for two cases, Human vs. Chimpanzee and Wheat subgenomes B and D, where we do not apply any filtering at all and run Simulated Annealing only once. These two cases also have the constraint of only being able to swap positions belonging to the same genome, that is, to achieve basketball views, whereas the others produce bubble views.

Table 5.1 shows the initial number of chromosomes, blocks, and crossings using the unfiltered version of each genome. Then, it shows the average times in seconds from the three runs, including the decluttering time (i.e. Simulated Annealing plus the manual time), and the manual intervention time alone. Finally, it shows the average number of crossings after the filtering gets reversed, and how much each genome improved in terms of the initial and final number of crossings. When taking into account these five genomes, on average, AccuSyn takes 60.92 seconds in total to declutter, including 39 seconds of manual intervention time, and there is an average decrease of block crossings (*improvement*) of 77%. Note that the Camelina genome as shown in Figure 5.1(a) on page 47 is a slightly filtered version that achieves a bubble view, but in reality this genome has many low-density links passing through its center (*outliers*) when being unfiltered, which is why there are so many crossings even after 31% of improvement.

In particular, four out of five genomes indicate at least 83% improvement, showing the overall success of our decluttering process. More importantly, a detangling job that could have taken hours to do completely manually and without guidance before AccuSyn, now takes around a minute. As you can tell, for any view to get completely decluttered, our process relies on human interventions. However, once running Simulated Annealing at least twice, it is very easy for the human intuition to see where each chromosome should go. Our algorithm stands out for creating a near-optimal way towards decluttering any view.

6 Conclusions and Future work

6.1 Conclusions

Genomic visualizations are playing an important role augmenting human interpretation and judgment when analyzing complex genomic data. To meet this goal, the contributions of this thesis were twofold. Our first contribution was AccuSyn, a web-based synteny browser, capable of producing highly customizable plots using two side-by-side views: the genome and the block view. These views encoded conserved links at three levels: the genome and chromosome level using a Circos plot, and individual block level using a bipartite bar plot. AccuSyn provides a high-level of interactivity and performance, by using the latest web technologies to produce a modern and functional website that offers a rich experience to the end users. The most significant considerations for the implementation of AccuSyn have been provided in chapter 4 of this thesis.

Our second contribution is a feature of AccuSyn and was the main focus of this thesis. We applied the idea of crossing number in graph theory to synteny visualizations. That is, we addressed the problem of visualizing conserved syntenic blocks with a minimal number of link crossings, by automating the creation and decluttering of synteny Circos plots. We showed how we cast this problem onto the Simulated Annealing metaheuristic, by implementing swapping and flipping decluttering operations and a unique technique to detect and minimize syntenic block crossings. Our novel use of Simulated Annealing is the first approach to automate the decluttering of a genome while visualizing as much genomic data as possible and allowing user interventions as a human-in-the-loop approach. All the algorithm details have been presented in chapter 3 of this thesis, along with a discussion of our implementation and a table of decluttering results in chapter 5.

This thesis contributed to the synteny interpretation bottleneck by automating the process of decluttering conserved synteny plots, thus highlighting the most important relationships for the user. Simulated Annealing adapted well to our problem helping in achieving a controlled stochastic approach, where users could jump in to contribute towards decluttering any genome. This exploratory research involved a lateral integration of several areas of knowledge: genetics, information visualization, and two areas of artificial intelligence: heuristic search and human-in-the-loop. Our implementation shows the potential that, in general, decluttering algorithms have in assisting the understanding of synteny plots. Thanks to AccuSyn, biologists are now able to quickly get an accurate representation of the conserved synteny within one or between two or more species, providing insights into the understanding of the evolutionary history between them.

6.2 Future work

Given the interdisciplinary and exploratory nature of this thesis, the decluttering work presented was by no means exhaustive. Our work has provided an excellent starting point given that the algorithm, being a metaheuristic, did not "know" anything about identifying genetic patterns. We just understood enough genetics to be able to apply Simulated Annealing to any synteny dataset. Therefore, we see the potential for other optimization algorithms or techniques to be a better fit as computer scientists obtain a deeper understanding of genetic sequences. We now present a few recommendations that might improve AccuSyn and its decluttering abilities:

- **Investigate ways to improve the decluttering results using Simulated Annealing:**

  For the algorithm to find a neighboring solution, we adapted the flipping and swapping decluttering operations to randomly choose one and two chromosomes, respectively. However, in reality, these operations can be varied to affect any number of chromosomes, creating a potential area of work to examine changes to the behavior of these operations that can enhance the end result.

  The same can be said about the metric used to measure the goodness of a solution (our objective function). It would be good to examine simpler metrics in terms of calculation, such as: a probabilistic metric, which could measure the probability of a solution being near the optimum; or measuring the Euclidean distance between the endpoints of block connections, although for the purpose of this calculation we would need to transform our Circos plots to linear-equivalent plots.

  Given that Simulated Annealing has not been part of the plant genetics toolkit (to the best of our knowledge), a straightforward recommendation is to find an automatic way to assign the default algorithm parameters in this domain. This would mean getting defaults that are good enough for a specific loaded dataset, not necessarily based on the number of block collisions of each layout instance. For example, the more block connections a dataset has, the fewer iterations that the defaults would need to contemplate, in general, for any of its layouts.

  We mention in chapter 5 on page 46 that one limitation of Simulated Annealing is that it takes great steps in the first few runs, but it seems to get stuck (i.e. does not find better solutions) when trying to improve further, for instance, when decluttering single crossings. A good area of future research is to look for endgame strategies to find ways to deal with those crossings that are harder for Simulated Annealing to find. Similarly, we can improve our beginning strategies when looking for a head start. For example, it might be beneficial to find chromosomes that have stronger relations (i.e. high-density links) and bringing them closer before running Simulated Annealing. Exploring further these two strategies would help in alleviating the amount of human effort when using manual interventions.

  We also mention in chapter 3 on page 36 that one manual technique we used was filtering the connections by block length until the crossing number "seemed" manageable, then running the Simulated Annealing algorithm, and finally adding some of the weak connections back in, and repeating this process until a satisfactory result was obtained. We conjecture that it would be possible to devise an algorithm that does this automatically and quickly, but we speculate that this initially is going to require a lot of empirical work and there are presently relatively few datasets to generalize from.

- **Use of machine learning techniques to improve clustering and human-in-the-loop:**

  Having a human-in-the-loop approach towards decluttering was an essential part of achieving plots with elegant appearance, like the Wheat genome presented in this thesis. This is why a promising area of future research is to consider pattern recognition techniques that could help in arriving at these "bubble" or "basketball" views automatically based on the copies that a genome includes, if known, even approximately. We could even use human interventions, if possible, to further enhance this idea, as a new machine learning approach. This would require to test with as many datasets as possible to be able to create a model that could predict accurately these types of clear synteny streams.

  For "bubble" views, k-means clustering could be used to group "bubbles" by distinguishing clusters from outliers, i.e. block connections that do not follow the "bubble" trends. The following would be a rough idea for the algorithm to be run on a likely "bubble" layout: first, change the representation of all block connections from Circos chords to straight lines; then, find the midpoints of all the line segments and use them as starting points, along with the centre point, given that the midpoints of outliers are very close to the center; finally, run k-means to find the clusters. This would quickly detect the groups around the initial points and the center, given that the midpoints would be close together, which would help in distinguishing "bubbles" from outliers, to be able to color them differently and focus on the most meaningful patterns. Note this assumes an ideal starting point, but even so, it shows its potential as future research because just identifying outliers would help in deciding when a decluttering solution is good enough.

- **Represent syntenic relations differently:**

  We could extend our current use of Circos plots to partitioning the genome into multiple Circos plots subsets, i.e. disentangling the subsets of chromosomes that are currently connected. For example, if we are visualizing a genome with five chromosomes, from "chr1" to "chr5," and "chr1" is connected with both "chr2" and "chr3," but "chr4" is connected only with "chr5," then we could show these subsets as two Circos views, instead of one. This could improve our idea of disentanglement, especially in genomes that show multiple copies of their structure.

  Moreover, given that we have a graph as the underlying structure of our synteny plots, we could use force-directed graphs to represent synteny. That is, having chromosomes or genes as vertices and conserved blocks as edges, and assigning forces to them that could simulate the amount of conserved synteny between the vertices. This type of plot might provide new insights revealing how messy or clear syntenic connections are, which in consequence could help with our decluttering problem, given how related vertices would be drawn close to each other.

- **Add more useful features to AccuSyn:**

  For the genome view, we could add the ability to zoom in and out, as in the block view, so users can enlarge block connections and visualize them better. Similarly, we could add scales to the genome view, to show the coordinate information for each chromosome based on the size of the genome, helping users obtain the approximate size of syntenic blocks without entering into the block view. Additionally, we could improve AccuSyn's filtering abilities, by adding options to filter by block identifier, match score, e-value, or gene identifier, to quickly show all the syntenic blocks that include a specific gene. Finally, we could improve the block view by adding the orientation of the genes, which could be represented as oriented glyphs at their specific locations. This change would require adding an extra column to the GFF file with the orientation of each gene, having the advantage of getting an extra piece of information for each block.

References

[1] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* John Wiley & Sons, Inc., New York, NY, USA, 1989.

[2] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] Jeong-Ho Baek, Junah Kim, Chang-Kug Kim, Seong-Han Sohn, Dongsu Choi, Milind B Ratnaparkhe, Do-Wan Kim, and Tae-Ho Lee. MultiSyn: A Webtool for Multiple Synteny Detection and Visualization of User's Sequence of Interest Compared to Public Plant Species. *Evolutionary Bioinformatics*, 12:EBO–S40009, 2016.

[4] Venkat Bandi. SynVisio: Synteny Browser. `https://synvisio.usask.ca`, 2018. Human-Computer Interaction Lab. Department of Computer Science. University of Saskatchewan. Retrieved June 11, 2019.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec 2011.

[6] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, Jan 1985.

[7] Feng Cheng, Jian Wu, and Xiaowu Wang. Genome triplication drove the diversification of Brassica plants. *Horticulture Research*, 1:14024, 2014.

[8] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011):931–945, 2004.

[9] Ke-Lin Du and M. N. S. Swamy. *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature.* Birkhäuser Basel, 1st edition, 2016.

[10] Hampton Catlin et al. Sass: Syntactically Awesome Style Sheets. `https://sass-lang.com`. Retrieved June 25, 2019.

[11] István Fáry. On straight-line representation of planar graphs. *Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

[12] Kenneth Ford, Patrick Hayes, Clark Glymour, and James Allen. Cognitive Orthoses: Toward Human-Centered AI. *AI Magazine*, 36:5–8, Dec 2015.

[13] Stephanie Forrest and Melanie Mitchell. Relative Building-Block Fitness and the Building-Block Hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms*, volume 2 of *Foundations of Genetic Algorithms*, pages 109 – 126. Elsevier, 1993.

[14] Mozilla Foundation. JavaScript. `https://developer.mozilla.org/en-US/docs/Web/JavaScript`. Mozilla Developer Network. Retrieved June 25, 2019.

[15] M. Garey and D. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[16] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov 1984.

[17] L. P. P. P. Van Ginneken and R. H. J. M. Otten. An inner loop criterion for simulated annealing. *Physics Letters A*, 130(8):429 – 435, 1988.

[18] Nicolas Girault. CircosJS. `https://github.com/nicgirault/circosJS`, Nov 2014. GitHub repository.

[19] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533 – 549, 1986.

[20] Fred Glover. Tabu Search—Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[21] Fred Glover. Tabu Search—Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[22] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *ArXiv E-prints*, page arXiv:1312.6082, Dec 2013.

[23] Manfred G. Grabherr, Pamela Russell, Miriah Meyer, Evan Mauceli, Jessica Alföldi, Federica Di Palma, and Kerstin Lindblad-Toh. Genome-wide synteny through highly sensitive sequence alignment: Satsuma. *Bioinformatics*, 26(9):1145–1151, 2010.

[24] Guy Gugliotta. Deciphering Old Texts, One Woozy, Curvy Word at a Time. `https://www.nytimes.com/2011/03/29/science/29recaptcha.html`, March 2011. The New York Times. Retrieved May 10, 2019.

[25] Brian J. Haas, Arthur L. Delcher, Jennifer R. Wortman, and Steven L. Salzberg. DAGchainer: a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–3646, July 2004.

[26] Maximilian Haeussler, Ann S. Zweig, Cath Tyner, Matthew L. Speir, Kate R. Rosenbloom, Brian J. Raney, Christopher M. Lee, Brian T. Lee, Angie S. Hinrichs, Jairo Navarro Gonzalez, David Gibson, Mark Diekhans, Hiram Clawson, Jonathan Casper, Galt P. Barber, David Haussler, Robert M. Kuhn, and W. James Kent. The UCSC Genome Browser database: 2019 update. *Nucleic Acids Research*, 47(D1):D853–D858, Nov 2018.

[27] Bruce Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[28] Mark Harrower and Cynthia A. Brewer. ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[29] Leland Hartwell, Leroy Hood, Michael Goldberg, Ann Reynolds, and Lee Silver. *Genetics: From Genes to Genomes*. McGraw-Hill, 4th edition, 2011.

[30] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[31] Andreas Holzinger. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, June 2016.

[32] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

[33] Facebook Inc. React – A JavaScript library for building user interfaces. `https://reactjs.org`. Facebook Open Source. Retrieved June 25, 2019.

[34] Twitter Inc. Bootstrap – The most popular HTML, CSS, and JS library in the world. `https://getbootstrap.com`. Retrieved June 25, 2019.

[35] L. Ingber. Very fast simulated re-annealing. *Mathematical and Computer Modelling*, 12(8):967 – 973, 1989.

[36] L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29 – 57, 1993.

[37] The Arabidopsis Genome Initiative. Analysis of the genome sequence of the flowering plant *Arabidopsis thaliana*. *Nature*, 408(6814):796–815, 2000.

[38] The International Wheat Genome Sequencing Consortium (IWGSC). Shifting the limits in wheat research and breeding using a fully annotated reference genome. *Science*, 361(6403), 2018.

[39] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by Simulated Annealing: an experimental evaluation; Part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[40] Sateesh Kagale, Chushin Koh, John Nixon, Venkatesh Bollina, Wayne E. Clarke, Reetu Tuteja, Charles Spillane, Stephen J. Robinson, Matthew G. Links, Carling Clarke, Erin E. Higgins, Terry Huebert, Andrew G. Sharpe, and Isobel A. P. Parkin. The emerging biofuel crop *Camelina sativa* retains a highly undifferentiated hexaploid genome structure. *Nature Communications*, 5:3706, April 2014.

[41] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[42] Liisa B. Koski and G. Brian Golding. The closest BLAST hit is often not the nearest neighbor. *Journal of Molecular Evolution*, 52(6):540–542, 2001.

[43] Martin Krzywinski, Inanc Birol, Steven J. M. Jones, and Marco A. Marra. Hive plots—rational approach to visualizing networks. *Briefings in Bioinformatics*, 13(5):627–644, Dec 2011.

[44] Martin Krzywinski, Jacqueline Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: an information aesthetic for comparative genomics. *Genome Research*, 19(9):1639–1645, 2009.

[45] Jongin Lee, Woon-young Hong, Minah Cho, Mikang Sim, Daehwan Lee, Younhee Ko, and Jaebum Kim. Synteny Portal: a web-based application portal for synteny block analysis. *Nucleic Acids Research*, 44(W1):W35–W40, 2016.

[46] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[47] Stephen Makonin, Daniel McVeigh, Wolfgang Stuerzlinger, Khoa Tran, and Fred Popowich. Mixed-Initiative for Big Data: The Intersection of Human + Visual Analytics + Prediction. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 1427–1436. IEEE, 2016.

[48] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[49] M. Meyer, T. Munzner, and H. Pfister. MizBee: A Multiscale Synteny Browser. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):897–904, Nov 2009.

[50] Heinz Mühlenbein. Evolution in Time and Space – The Parallel Genetic Algorithm. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1 of *Foundations of Genetic Algorithms*, pages 316 – 337. Elsevier, 1991.

[51] Cydney B. Nielsen, Michael Cantor, Inna Dubchak, David Gordon, and Ting Wang. Visualizing genomes: techniques and challenges. *Nature Methods*, 7(3s):S5 – S15, March 2010.

[52] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373–8385, Oct 1998.

[53] I. A. P. Parkin, A. G. Sharpe, and D. J. Lydiate. Patterns of genome duplication within the Brassica napus genome. *Genome*, 46(2):291–303, 2003.

[54] Eberhard Passarge, Bernhard Horsthemke, and Rosann A. Farber. Incorrect use of the term synteny. *Nature Genetics*, 23(4):387, 1999.

[55] William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[56] Alexander Pertsemlidis and John W. Fondon III. Having a BLAST with bioinformatics (and avoiding BLASTphemy). *Genome Biology*, 2(10):reviews2002–1, 2001.

[57] Pavel Pevzner and Glenn Tesler. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Research*, 13(1):37–45, 2003.

[58] Sebastian Proost, Jan Fostier, Dieter De Witte, Bart Dhoedt, Piet Demeester, Yves Van de Peer, and Klaas Vandepoele. i-ADHoRe 3.0—fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic Acids Research*, 40(2):e11–e11, Nov 2011.

[59] Kashi V. Revanna, Daniel Munro, Alvin Gao, Chi-Chen Chiu, Anil Pathak, and Qunfeng Dong. A web-based multi-genome synteny viewer for customized data. *BMC Bioinformatics*, 13(1):190, 2012.

[60] Marc H. J. Romanycia and Francis Jeffry Pelletier. What is a heuristic? *Computational Intelligence*, 1(1):47–58, 1985.

[61] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[62] Marcus Schaefer. The Graph Crossing Number and its Variants: A Survey. *The Electronic Journal of Combinatorics*, 1000:21–22, 2013.

[63] F. Shahrokhi, O. Sýkora, L. Székely, and I. Vrto. On Bipartite Drawings and the Linear Arrangement Problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.

[64] Kim Sharp and Franz Matschinsky. Translation of Ludwig Boltzmann's Paper "On the Relationship between the Second Fundamental Theorem of the Mechanical Theory of Heat and Probability Calculations Regarding the Conditions for Thermal Equilibrium" Sitzungberichte der Kaiserlichen Akademie der Wissenschaften. Mathematisch-Naturwissen Classe. Abt. II, LXXVI 1877, pp 373-435 (Wien. Ber. 1877, 76:373-435). Reprinted in Wiss. Abhandlungen, Vol. II, reprint 42, p. 164-223, Barth, Leipzig, 1909. *Entropy*, 17(4):1971–2009, 2015.

[65] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, Sep 1996.

[66] Temple F. Smith and Michael S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[67] Carol Soderlund, Matthew Bomhoff, and William M. Nelson. SyMAP v3.4: a turnkey synteny system with application to plant genomes. *Nucleic Acids Research*, 39(10):e68–e68, 2011.

[68] Kenneth Sörensen. Metaheuristics–the metaphor exposed. *International Transactions in Operational Research*, 22:3–18, 2015.

[69] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big Data: Astronomical or Genomical? *PLoS Biology*, 13(7):e1002195, 2015.

[70] Haibao Tang, Matthew D Bomhoff, Evan Briones, Liangsheng Zhang, James C Schnable, and Eric Lyons. SynFind: Compiling Syntenic Regions across Any Set of Genomes on Demand. *Genome Biology and Evolution*, 7(12):3286–3298, 2015.

[71] Haibao Tang, Xiyin Wang, John E. Bowers, Ray Ming, Maqsudul Alam, and Andrew H. Paterson. Unraveling ancient hexaploidy through multiply-aligned angiosperm gene maps. *Genome Research*, 18(12):1944–1954, 2008.

[72] Constantino Tsallis and Daniel A. Stariolo. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1):395 – 406, 1996.

[73] Daniel Veltri, Martha Malapi Wight, and Jo Anne Crouch. SimpleSynteny: a web-based tool for visualization of microsynteny across multiple species. *Nucleic Acids Research*, 44(W1):W41–W45, 2016.

[74] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, 2008.

[75] World Wide Web Consortium (W3C). Cascading Style Sheets (CSS). `https://www.w3.org/Style/CSS/`. Retrieved June 25, 2019.

[76] World Wide Web Consortium (W3C). HTML5. `https://www.w3.org/html/`. Retrieved June 25, 2019.

[77] Yupeng Wang, Haibao Tang, Jeremy D. DeBarry, Xu Tan, Jingping Li, Xiyin Wang, Tae-ho Lee, Huizhe Jin, Barry Marler, Hui Guo, Jessica C. Kissinger, and Andrew H. Paterson. MCScanX: a toolkit for detection and evolutionary analysis of gene synteny and collinearity. *Nucleic Acids Research*, 40(7):e49–e49, Jan 2012.

[78] Kenneth H. Wolfe. Yesterday's polyploids and the mystery of diploidization. *Nature Reviews Genetics*, 2(5):333, 2001.

[79] Xinghuo Zeng, Matthew J. Nesbitt, Jian Pei, Ke Wang, Ismael A. Vergara, and Nansheng Chen. OrthoCluster: A New Tool for Mining Synteny Blocks and Applications in Comparative Genomics. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, pages 656–667. ACM, 2008.