Beautiful Genomes

Abstract

We describe work arising from a practical collaboration between computer scientists and biologists to visualize synteny (similarity) in genomes as part of a larger project in REDACTED. The biologists wanted an interactive synteny visualization software tool that produced output similar to MizBee, but also automated the generation of the synteny diagram, a current bottleneck in the process. Another requirement was that the tool’s interface be easily understood by non-computer experts.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Data Visualization, Bioinformatics

1. Introduction

Advances in genome sequencing and mapping have enabled deep comparisons of the genetic structures of increasingly complex species, including humans. Species believed to have diverged from common ancestors tens of millions of years ago (cows, humans, mice) share significant features. A feature of interest may be a gene, or a larger syntenic block of contiguous features. These genetic linkages, collectively called synteny, in turn confirm the divergence of species. These syntenic relations not only answer evolutionary questions and illuminate evolutionary events, they allow comparative insights into the functioning of organisms.

The first plant to be completely sequenced was Arabidopsis thaliana [HIG04] in 2000. The sequencing of this flowering plant with only five chromosomes and 27,000 genes by the Arabidopsis Genome Initiative posed a significant challenge in its day, and was hailed as a major milestone for biology. But identifying syntenic blocks among these 27,000 genes, and more significantly, understanding patterns therein posed computing challenges at that time. Since then, the sequencing process has been simplified and sped up, resulting in larger genomes to visualize.

Our software derived much inspiration from MizBee [MMP09], the first synteny browser to provide linked views of syntenic relations across the entire genome, but also between chromosomes and blocks, capturing the visualization principle of making data sets coherent and revealing information at several levels of detail [CKB09]. That pioneering work claimed four major contributions. Firstly, it clearly characterized the problem domain, and secondly, gave a taxonomic analysis of the visual encodings suitable for this data. The third was the physical browser, with which ours shares several similarities. It also presented case studies resulting from discussions with MizBee users.

At about that time, Neilson et al [NCD*10] stated that data analysis was replacing data generation as the rate-limiting step in genomic studies. Genomics was just one of many areas experiencing the challenges of understanding large quantities of data, the volumes of which make it difficult to gain a “big picture” of the result.

Figure 1: MizBee genome visualizer (requesting permission)

The designers of MizBee [MMP09] had hinted at some of those challenges in the context of depicting relationships within and among genomes. They used the edge bundling technique to reduce visual clutter by bundling connections from neighbouring blocks. It also highlighted a case study where a collaborator used the viewer to develop an algorithm to find conserved syntenic blocks within stickleback and pufferfish genomes [GRM*10]. The collaborator expressed initial disappointment at the cluttered data. The algorithm was subsequently refined to show the most important relations, suggesting a need for decluttering functionality within the viewer.

Many genome browsers followed MizBee (see REDACTED for a summary) but to the best of our
knowledge, ours is the first to provide users with real-time software tools that automate the decluttering process, both by providing an animated tool that lets the human manually disentangle the connections, along with algorithms that independently tackle decluttering, and are organized so as to facilitate a mixed-initiative approach, as the algorithm occasionally seems to get stuck, and benefits from an occasional human assist. Additionally, this leaves room for subjective judgement if it is important to favor a particular visualization, e.g., grouping of chromosomes.

This software resulted from a collaboration between computer scientists and biologists as part of a larger ongoing project, which this constrained the set of tools that could be used. Another constraint was the biologists’ wish to deploy the tool in a web browser so that it could be used widely. There was also an interesting UI challenge. On the one hand, many of the biologists expected to use this tool were not computer experts and wanted the software to offer an “out of the box” experience. On the other hand, most of the target users were far more sophisticated computer users than typical lay people, and we had to balance how much of the internal workings we wanted to give users access to [CGS*14].

Another challenge was communication of requirements. The computer scientists initially had minimal background in genomics and initial conversations were at a syntactic level. This changed quickly, however, and the groups had many back and forth consultations.

The next section describes the software tools used in this project. Many were chosen because they were integrated into other tools in the larger project.

2. Background

Part of the challenge in finding syntenic relations is the degree of error (uncertainty or noise) we choose to accept. Chromosomal rearrangements (deletions, inversions, translocations) occur over time in error, even though these may accumulate to give an organism an advantage. As a result, similar blocks are not always identical, and numerous algorithms have been suggested for detecting ‘acceptable’ syntenic links, and all of these algorithms contain parameters that must be tuned. This raises many issues about the integrity of resulting data, especially data produced as techniques were emerging.

The larger project with which the work was associated was already using a software tool called MCScanX [WTD*12] for detecting syntenic blocks using both a Genomic Feature Format (GFF) file and a BLASTp file, which contains all the sequences of interest. We do not discuss sequence alignment algorithms or file formats here, except to say that BLAST (Basic Local Alignment Search Tool) [AGM*90] appears to be the most popular tool for finding similar blocks. The outputs contain information about matching block pairs, including their location and measurements of alignment strength. We refer the reader to [WTD*12] for details. As our software has been built modularly, it could accommodate other file formats or syntenic link detection algorithms with little trouble provided the fundamental representation doesn’t change.

MCScanX is also capable of displaying its results using several plot types, including linear and circular plots arranged in lines or around a circle. It also used dot plots and bar plots. Our part of the project concentrated on the use of circular plots, where genomes are arranged around a circle using ribbons to show syntenic linkages compactly over the whole genome.

MCScanX’s circular plots are created with Circos (Krzywinski et al., 2011), a command-line non-interactive viewer that generates such circular layouts. Figure 2 shows a Circos diagram.

Figure 2: Circos diagram (Requesting Permission)

Another requirement was that our solution be a web application accessible worldwide. Thus it had to be based on the three common web languages: JavaScript, HTML, and CSS3 to provide interactivity, structure, and styling to the site, respectively. However, besides native Javascript, our solution uses three JavaScript libraries to create the genome and the block view along with the configuration panel: D3.js [BOH11], CircosJS [Gir14], and React.js [Fac19]. D3.js (Data-Driven Documents) provides 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 also uses Scalable Vector Graphics (SVG) to create resolution-independent visualizations.

CircosJS is the D3.js version of the original Circos software and was the starting point for creating interactive circular genome views, which we extended to include the decluttering tools discussed later. React.js helped in creating dynamic user interfaces, by efficiently updating and controlling the DOM. Circos leverages the huge amount of genomic data by displaying additional data in multiple tracks, such as heatmaps, histograms, lines, and scatter plots. It adds value by reducing the implicit difficulties of visualizing large-scale genomic data. Circos takes input files similar to the Genome Feature Format (GFF) file format and outputs images in PNG or SVG formats, and its plots are frequently the norm when sharing new syteny results to those studying comparative genomics, e.g. (REDACTED). Circos was the first viewer to use circular plots to visualize similar relationships, an ideal technique for getting a complete overview of a genome. As its authors describe it, Circos presents images that are “clear and informative to the investigator and attractive and compelling to the general public” [KSB*09].
For a survey of synteny browsers including mGSV (multi-Genome Synteny Viewer), SimpleSynteny, MultiSyn, Synteny Portal, and SynVisio, see (REDACTED). All of these extend the kinds of information that can be displayed on circular plots. For new work on automated generation of visualization other than circular styles, see (REDACTED).

3. Introducing Interactivity

3.1 Genome View

The first goal for creating our synteny viewer was implementing an interactive version of MizBee’s [MMP09] genome view (the circular display in Figure 1). MizBee defined a taxonomy of layouts for synteny diagrams including contiguous linear layouts and circular layouts. Circular layouts could either separate source and destination chromosomes into separate rings or combine them into a single ring. Figure 1 shows MizBee’s source chromosomes along the outside. The inner ring shows the destination chromosomes in grey arranged around a single (coloured) source chromosome also highlighted on the outer ring.

Our implementation follows the seven tasks of the Visual Information Seeking Mantra, a complete framework for designing Information Visualization applications [Sch96], 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 by looking at genomes or chromosomes as arcs in a single circle, instead of multiple stacked horizontal lines [NCD10].

As our goal was an interactive “scratchpad” for biologists to experiment with layouts for study and publication, we chose to implement the initial genome view as a single ring containing all chromosomes and displaying all possible one-to-many block connections. As Figure 3 shows, the complexity of a complete diagram of some of the recently sequenced genomes (in this case, Chinese Spring Wheat) suggests that decluttering a complete diagram may be daunting to a user. However, the user may select just one chromosome in the genome view, and see an overview all the conserved block connections the selected chromosome has with all the others. The user may also select multiple chromosomes, that is, the user may hide most of the chromosomes to make observation of the many-to-many block connections between selected chromosomes easier. The user can then incrementally add missing chromosomes back in as partial structures become understandable. Another step in improving the clarity of the visualization is the addition of colour.

For colouring, our software follows the specifications of ColorBrewer [HB03], a tool for choosing 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. Figure 4 shows that adding colour to the connections considerably declutters the drawing.
Once the diagram is coloured and filtered, the user can then drag chromosomes around the genome to reorganize them arbitrarily as suits the information they wish to display.

By default, our software displays chromosomes sequentially in alphabetical ascending order. Different users may have different ways for arranging chromosomes; for our purposes, the primary task was disentangling the connections. Figure 6 illustrates how easy manual disentangling is in our software.

The dragging operation works consistently with other dragging tools, and intuitively for this setting. When the user clicks a chromosome, the chromosome is cloned, and its label and connections are highlighted as the clicked chromosome is dragged around the genome, its connections following it, while the pointer button is held down. Figure 6 shows that a place is “held” while the chromosome moves both to eliminate distracting movement, and to show the user where to replace the chromosome given a change of mind. When the user wishes to “drop” the chromosome into a new position, the user just lets go of the pointer button triggering a sequential animation that translates the chromosome positions one by one until the selected chromosome can slide over into the new place. This minimizes movement in the image as the selected object is moved.

A related operation is flipping. In many MCScanX outputs, chromosomes were homologous except that the order of blocks in some chromosome was flipped end to end, resulting in a “tangle” that could not be eliminated by dragging. Figure 7 illustrates the problem.

The user can flip the offending chromosomes with a right click of the pointer, resulting in Figure 8.
A close look at Figure 8 shows that even after dragging and flipping, there remain unexplained stray blocks. However, the overview image clearly suggests the presence of two sets of homologous chromosomes.

4. Decluttering of Synteny Diagrams

The interactive tools of the previous section are not sufficient to produce uncluttered synteny diagrams quickly. This presented us with another opportunity/challenge: find a way to cast this as an optimization problem that could be solved heuristically. Combinatorial enumeration might work at this time for simple plants like Arabidopsis, but with the passage of time biologists are studying larger and larger genomes.

This was a double-barreled challenge. The first was finding a feasible objective function given the constraints of the implementation, including the design of existing software. However, optimization algorithms may contain parameters and stopping criteria that require that users have some understanding of the algorithm. On the one hand, we could expect that our users would be more comfortable with computer technology, including the general idea of optimization, than a lay person. On the other hand, the computer scientists on the team were on the ground floor regarding the biology and recognized there could be much variance amongst users whose background was mostly in biology. The second issues was deciding how much access to the algorithm’s parameters users should have.

The algorithm chosen was simulated annealing (SA) [KGV83], which originated in management science and later became part of the canon of search algorithms used in heuristic artificial intelligence, and is based on assumptions similar to those underlying genetic algorithms.

Put simply, given some objective function that measures the “goodness” of the solution, SA scrambles a solution incrementally and (somewhat) randomly. This is called the annealing schedule. In our software, it first “rolls the dice” to select two chromosomes to swap. If swapping them improves the solution, SA continues. If the goodness of the solution worsens, SA “rolls the dice” again and accepts the poorer solution with probability $p(t)$, where $t$ is time, and $p(t)$ decreases with time. It behaves like a greedy algorithm, until it gets stuck in a local minimum. At that point, this probabilistic trick will make it eventually give up and go back to the drawing board, and approach the problem from a different direction. As time passes, it is less and like likely to accept a bad solution. SA typically uses a fixed number of iterations derived from the annealing schedule, so it will always stop.

Simulating annealing has two key parameters, “temperature” and “cooling rate”, and possibly a third, maximum run time. The main question is whether to make these parameters accessible from the control panel. The names of the first two parameters come from metallurgy and the process of heating metals then cooling them slowly to increase their strength. The computer scientists were also familiar with the concept of interface bloating discussed by McGrenere [MM90] and wondered whether giving access to SA’s internals would confuse a process we had worked hard to simplify. In the end, it was possible to create a table of parameters based on the size of the input that performed well in practice for all the datasets we worked with, and we placed the SA parameters behind “Advanced Features” as this introduced little risk.

We measured the “goodness” of the diagram by counting the number of edge crossings. Because of the complexity of the various Javascript packages, rather than dive into the package to dig out the formulas for the curves, we used the following approximation. All connections (the curved lines) were projected onto straight lines, which made intersections easy to compute. This isn’t perfect, because, among other things, Circos uses edge bundling to render the connections.

![Figure 9: Progress bar during search for new solution](image)

Following the Nielsen heuristics [NM90, AS94] we implemented a progress bar (Figure 9) to show the user the state of the algorithm. This addresses visibility of system status, and gives the user control. It was not difficult to compute an expected completion statistic after some initial runs.

When the decluttering algorithm runs, SA can take from a few seconds up to a few minutes to find a possible solution. The user is the one to decide whether a solution is worth waiting for, and we display progress to completion in real time to the user. Since decluttering cases are computed so quickly, it is impractical to render every rearrangement. Thus, as the algorithm runs we display the graph out of focus behind an animated progress bar, then use a simple algorithm to animate the chromosomes moving to their new locations one at a time.

5. Mixed-Initiative Decluttering

In practice, given this objective function and this annealing schedule, we found that simulated annealing quickly goes from bad to reasonable solutions, but struggles to go from a reasonable solution to the best one. Consider the degenerate case where a syntenic layout has two chromosomes across each other, and their connections cross all the other crossings in the graph. None of the other chromosomes cross each other. A common-sense solution at this point is to place these two chromosomes side by side, but the objective function and annealing schedule (swapping two nodes) we have described won’t do that. In the time allotted, the algorithm often gets to a point where it is mostly making the solution worse.

We conjectured that the algorithm gets stuck at reasonable but not great solutions because the “tangle” can only be fixed...
by a very specific series of moves, and the random selection annealing schedule runs out of time before it finds them.

Since this happened at a point when the solution was reasonable, it was often clear to the user which chromosomes should be moved to untangle the links. This suggested we design the software with a "mixed-initiative" perspective.

The idea of "mixed-initiative" (also known as "human-in-the-loop", "human-centered", "cognitive prosthetic") appears variously in the HCI and AI literature. Horvitz [Hor94] describes these as techniques that enhance users’ abilities to directly manipulate objects rather than developing interface agents that provide automation. In a similar vein, [FGA15] write about thinking about intelligent software as that which acts as a prosthetic to a human’s cognitive abilities rather than outperform a human autonomously. In any case, recent anecdotal experience with speech to text generation for example, suggests that there remain simple tasks at which humans outperform machines, and at certain points in a computation, a human will quickly intuit a solution that the machine would not.

This worked well in practice. During the course of a computation, the machine did well on the first pass, at which point the human would either run the algorithm again, or "break the log jam" by manually moving a few chromosomes before passing the problem back to the machine, with the human in the end often handling the "end game". Figures 10, 11, and 12 show the net results are very good.

**Figure 10: Chinese Spring Wheat Grouped**

Figure 10 shows the result that follows Figure 4 of Chinese Spring wheat, and the grouping that results. Figure 11 below shows the same graph unfiltered.

Figure 12 shows Chinese Spring Wheat unfiltered in a "dark mode", one of the myriad ways our software can stylize the image.

These images of Chinese Spring Wheat are the first images produced of the genome after it was sequenced. The final image shows additional tracks providing other statistical information, which are similar to those appearing in Figure 2. Additionally it has been recoloured to reflect the multipliod genome – it consists of three genomes that have combined during evolutionary events.
Most of the images presented in this paper feature multiploid organisms. We call these bubble diagrams, because the algorithm (by chance) eventually places related chromosomes from subgenomes together. However, the software has produced other interesting visualizations. Figure 13 shows a "basketball" diagram that emerges when grouping together just two of the wheat subgenomes.

Figure 13: Basketball view of Wheat subgenomes B and D

It is possible to force bubble diagrams to be rendered in the bubble style. Figure 14 shows the information in Figure 13 rendered as bubbles.

Figure 14: Bubble view of Wheat subgenomes B and D

It was interesting to see the result of plotting the human and chimpanzee genomes against each other (Figure 15).

Figure 15: Human and Chimpanzee

5.1 Simulated Annealing and Data Visualization

Why simulated annealing?

The crossing number problem, that is, determining the fewest number of edge crossings, is intractable \cite{GJ15}, leaving us to look for special cases and heuristics. Polynomial solutions exist when the graph is planar \cite{Far48} or bipartite \cite{Sch62} but perfect cases rarely arise as close inspection of the drawings herein will reveal. Thinking about our circular genome drawing graph-theoretically, it is more like a hypergraph than a graph, as moving a chromosome typically results in the movement of many blocks.

Another question that comes up is, why not look into Machine Learning, especially the recent developments in Deep Learning?

Both directions offer potential for future research. This exploratory investigation revealed clear patterns in synteny (at least in the genomes we looked at) that could guide search. However, given temporal constraints, investing in deep graph theoretic or machine learning techniques would create a software engineering issue. Simulated annealing can be expressed concisely in a few lines of code that can be maintained and extended easily enough by a good undergraduate student. This approach provided interesting and encouraging results that will lead the area in other directions.

6. Other Features

The present work mostly discusses the visualizations of the genome data and the challenges of building a user interface for an unusual task. Informally, we found that after some experience, it takes 15 minutes or so (depending on the genome) to produce an attractive diagram, which makes certain features of MizBee less necessary. For example, there is no need to provide a chromosome view and a genome view; these can be constructed independently. A block view similar to that of MizBee was implemented which required a separate rendering and manipulating techniques. Figure 16 shows a screen capture of a fullfeatured starting point of our software.
7. Discussion and Conclusions

During this exploratory work, the computer scientists were repeatedly getting requirements from the biologists, implementing them, and seeing which outcomes were worthwhile for users. While simulated annealing turned out to be a productive first step in this search, and the mixed-initiative approach worked even better, thoughts turned to more analytic ways of handling the problems the annealing algorithm struggled with. A good start would be looking at “endgame” strategies and seeing whether a small set of specialized heuristics could speed up the last leg of the process. For example, consider the degenerate case of a genome with 26 chromosomes, where 24 are untangled but for a single pair that cross all the others. Even if it guesses the two correct chromosomes, swapping them will make no difference. It will take a great deal of luck and time to find the sequence that repairs a problem that is obvious to a human.

It would be worthwhile to consider alternate annealing schedules. For instance, it may be beneficial to begin the computation by finding pairs of chromosomes with high density of links connecting them, and bringing them closer together before applying simulated annealing. Similarly, it may be possible to filter the graph heavily at the beginning, and then slowly add links back in by running SA for each new instance. Our feeling at this time is such an approach would require a good deal of empirical work and some heavy parameterization to work in the general case, but this seems promising.

Entrenching this empirical understanding in a simulated annealing algorithm may be effective. However, the time already invested in tuning this software does suggest that there may be ways to find syntenic diagrams that are driven by our deep knowledge. Games like checkers and chess were originally driven by comparatively crude heuristics, and with time, but top performers are exploiting deep understanding of the game and its history. Over the long term, the same may apply in this domain. As well, the genomic datasets of particular interest to the biologists at this time exhibited a high degree of multiploidy, which may have skewed perceptions of the domain by those developing the software.

In conclusion, to the best of our knowledge, ours is the first approach to automatic genome decluttering. It also allows user interventions and makes it possible to visualize a huge amount of data all at once. The software has been presented at various relevant symposia and has been deployed on the web where it has attracted interest. (DETAILS REDACTED)

NOTE TO REVIEWERS: The name of the software and related projects have been omitted by way of compliance with the double-blind submission process.

References


