



algorithm, ABySS (Assembly By Short Sequences) [24], the visual encoding introduced here has the potential to generalize to other DNA sequence display problems.

Our visualization has been adopted by a number of genome analysts and subjected to iterative design based on their feedback. We discuss details of these design refinements, together with descriptions of how our visualization tool was used in different analysis tasks. This tool represents a significant contribution as it replaces laborious manual data-integration. Through discussions with users, we have identified additional applications not considered in our initial design process, highlighting the generality of our visual encoding of DNA sequences.

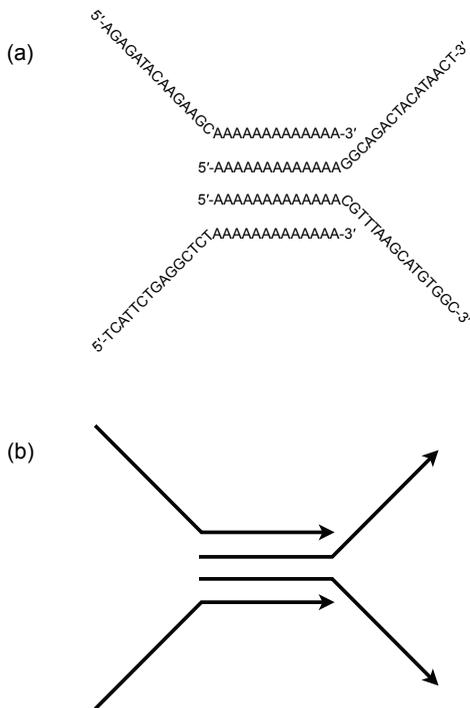


Fig. 2. Example of a path ambiguity in a genome assembly resulting from a repetitive A-rich sequence shown as DNA sequences (a) and as an arrow diagram (b).

## 2 ASSEMBLING A GENOME

The raw output of a genome sequencing pipeline is a collection of short sequencing reads. One of the core operations of a *de novo* genome assembly algorithm is to find overlaps between reads. Sequencing technology has undergone a revolution in recent years and next-generation techniques produce much shorter reads (25-100 nt compared to 500-1000 nt), however they can produce several orders of magnitude more data for the same cost. A single sequencing reaction currently produces millions of short reads. Standard overlap search algorithms are not optimized for this very large number of short sequences, and thus a new generation of assembly algorithms has emerged.

Our in-house assembly algorithm, ABySS [24], addresses the overlap search problem by representing DNA sequences as a de Bruijn graph, a notion pioneered by Pevzner *et al.* [21] and employed in other recently published genome assembly tools [4, 25, 6]. A de Bruijn graph is a directed graph that compactly represents a uniform overlap between sequences. All possible sub-strings of length  $k$  (termed  $k$ -mers) are generated from the sequencing reads. A vertex is created for each  $k$ -mer and edges join vertices when they overlap by  $k-1$  nucleotides. The assembly process consists of merging vertices that are unambiguously connected, and this computation can be distributed over a cluster of computer nodes. The resulting merged sequences are called contigs.

A shotgun sequencing experiment may produce many reads through a given nucleotide. Read coverage is defined as the average number of reads supporting a particular nucleotide at a given position in the assembly. Sequencing errors occur at a low rate (typically  $<1$  error in 100 nucleotides using the Illumina Genome Analyzer platform). Quality of the assembled sequence correlates with read coverage, as it is very unlikely that the same error will be observed multiple times at a given position. Thus the read coverage is computed for each contig in an assembly and constitutes a valuable metric during analysis.

Read pairing information offers additional information about an assembly. In a popular shotgun sequencing protocol, DNA molecules are extracted from cells and sheared into small fragments (roughly 200 nt). Each end of a given fragment is then used as a template to produce short sequencing reads. The pairing information between reads from either side of a given DNA fragment can be recorded. After contigs are assembled, the ABySS algorithm has a second phase which aligns pairs of sequencing reads to the contigs. The distance between each aligned read pair is recorded for cases where both reads match an existing contig, and these distances yield an estimate of the true fragment size distribution. Paired reads are particularly valuable when they align to different contigs as they can be used to infer the correct contig orderings and estimate inter-contig distances.

## 3 RELATED WORK

This work was motivated by the lack of adequate visualization tools for analyses of global genome assembly structure for new sequencing platforms. There are several powerful tools that provide local detailed views of read sequences and their alignments within contigs, such as Consed [9] and gap4 [3], however these tools provide no visual overview of an assembly as a whole. Phrapview [9], a companion tool to Consed, was among the earliest tools for examining contig connectivity, and it displays contigs in a parallel coordinates view. Connections between contigs, such as those supported by paired reads, are represented as connecting diagonal lines and color is used to disambiguate connections of different types. As with any parallel coordinates scheme, this view can become difficult to interpret when the number of connections is large. Even with relatively sparse data, it is challenging to visually follow the path through multiple parallel contigs and spot high-level structures such as cycles.

Hawkeye [22] provides an abstract overview of contigs as a collection of appropriately sized horizontal lines ordered along the  $x$ -axis. Details of paired read placement within a selected region are available in a linked view, and pairs are color coded to flag common assembly errors. This representation is powerful for some analysis tasks, such as recognizing read pairs separated by statistically extreme distances. However, visually tracing a path through a set of densely connected linearly ordered contigs can be difficult and connections to contigs in distant regions of the ordering are hard to identify.

ConPath [19] is the only visualization tool to our knowledge that displays contig connections as a graph. It arranges contigs, represented as vertices, using a grid layout with unambiguously ordered contigs arranged on the same row. Edges indicate ambiguous connections between contigs on different rows. The edges are forced to follow a grid making them very hard to distinguish, and neither vertex placement nor edge length provides an accurate representation of distance in the assembled sequence. ConPath generates non-interactive graphics that can be difficult to interpret when the number of ambiguous edges is large.

All existing assembly tools have very limited capability when it comes to displaying meta-data, such as gene annotations. Popular genome browsers, such as UCSC [18] and Ensembl [15], use a set of parallel horizontal tracks, one for each annotation type, that share a common genomic  $x$ -coordinate. However, all of these tools function on a reference genome sequence and are not designed to accommodate the connectivity information inherent in a *de novo* genome assembly. Neither Phrapview nor ConPath enable annotation display. Hawkeye, and a more recent tool EagleView [14], incorporate annotation tracks in parallel to the contig display, much like a genome browser. While useful on a local scale, these views fail to provide an easy to interpret

context for how the contigs connect to each other.

Discussions with genome analysts reveal that judgements about assembly errors are tackled one at a time using local information, and while they have metrics to guide them to the problematic regions in the assembly, they have no overview of the global problem. In addition, there is no way to easily examine the interconnections between contigs together with meta-data such as annotations. This paper addresses these problems with a visual encoding that emphasizes the connectivity information while integrating important data features into the representation which allows a global view of the assembly.

#### 4 DESIGN DECISIONS

Genome analysts require several types of information when examining a sequence assembly. These include: (1) information about which contigs overlap with each other, (2) contig orientation, (3) contig length, (4) contig sequence coverage, (5) paired read positions with distance estimates, (6) additional meta-data not used in the assembly such as gene annotations. Most of these data types are stored in distinct files with different file formats and no visualization tools exist to integrate all of them. We designed a visual encoding that captures all of these features in a single representation with the intent to eliminate the need for laborious cross-referencing between data sources. Inspired by the guidelines of Shneiderman [23], we built an interactive interface that provides an initial overview of the data set, while enabling subsequent focusing on regions of interest and access to details on demand.

##### 4.1 Contig Connectivity as Graphs

Graphs are good representations for displaying connectivity, however they are frequently misused in cases where the data structure is not the most interesting feature [8]. In our domain, one of the goals of an analyst is to decipher the structure of an assembled region and to disambiguate the paths between contigs. For complex structures, we found that analysts would frequently draw the corresponding graph of contig connections by hand in order to reason about the assembly structure. Given these observations, a graph is a very natural encoding.

As described in section 2, the ABySS algorithm uses a de Bruijn graph as an internal representation of sequence overlaps. This approach is not unique to ABySS [4, 25, 6]. Although some groups present static graphs of real genome assembly data for publication purposes [5], no interactive analysis tools exist for this data type. We initially applied the de Bruijn graph representation to our first design of an interactive assembly viewer. While the de Bruijn representation is well suited for the computation performed by the assembly algorithm, we quickly realized that this encoding is not optimal for visually reasoning about sequence space.

The ABySS assembly algorithm represents each contig as a vertex and uses edges to indicate an overlap of  $k - 1$  nt between two contigs. As shown in Figure 3, no strong visual correspondence exists between the de Bruijn graph representation (3a) and the true sequence space (3b). For example, the de Bruijn graph captures an overlap between the terminal end of contig 1 and the start of contig 4 with an edge. However, contig overlaps at start positions (e.g. between contigs 2 and 4) or at terminal ends (e.g. between contigs 1 and 3) are not as immediately obvious.

To obtain a representation that more directly maps onto sequence space, we inverted the graph encoding and represent contigs as edges and vertices as points of contig overlap. This change resulted in very positive feedback from analysts, and we argue that these revised graphs enable analysts to reason about the graph in a similar way to how they would reason about sequences. For example, in Figure 3c, it becomes easy to see that the source of the assembly ambiguity is a  $k - 1$  nt sequence shared by all four contigs.

We initialize a global overview of the graph in ABySS-Explorer using the Kamada-Kawai force directed graph layout algorithm [17] (see Section 5). Our interface supports interactive manipulations of the graph view such as translation, rotation, and zoom. The iterative layout algorithm that can be paused at any point and all vertices can be repositioned by hand if desired.

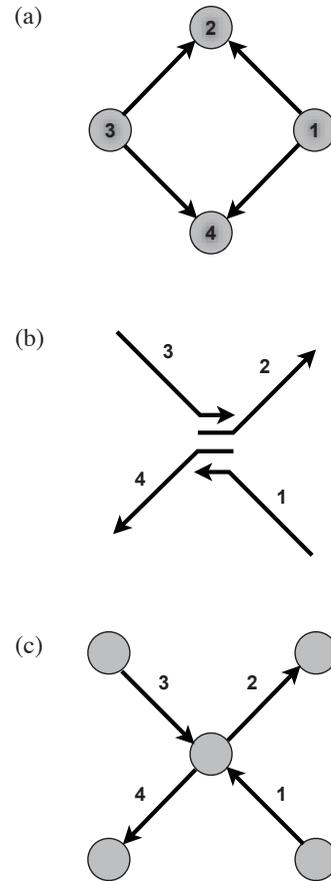


Fig. 3. Different representations of sequence contigs labelled 1-4. (a) Internal de Bruijn graph representation used by the ABySS assembly algorithm where contigs are represented as vertices and edges indicate a  $k-1$  nt overlap. (b) Arrow diagram of the corresponding sequences. (c) Graph representation used by ABySS-Explorer where contigs are represented as edges and vertices indicate a  $k-1$  nt overlap.

##### 4.2 DNA is Double-Stranded

At this point our graph representation captures a contig sequence as a single directed edge. However, DNA is double-stranded. The two DNA molecules (strands) are oriented in opposite directions (anti-parallel) and they are complementary (an A nucleotide in one corresponds to a T in the other, and a C nucleotide in one corresponds to a G in the other, and vice versa). We arbitrarily label the two strands “+” and “-”. As they carry the same information content, we choose one as the reference strand. Sequencing generates reads from both DNA strands, however we do not know which strand a read originates from *a priori*. In order to resolve the reference strand sequence, analysts require the ability to reverse complement any given contig.

One option for capturing the double-stranded nature of DNA is to ensure that the graph represents both a  $k$ -mer and its reverse complement as distinct vertices. The process of reverse-complementing a sequence then corresponds to reconnecting a contig edge to the appropriate vertex. However, this representation has unnecessary redundancy in that a given DNA sequence is encoded as two vertices, one for each strand. Controlling the layout of  $k$ -mer pairs also becomes cumbersome.

Instead, we make each vertex polar such that one pole corresponds to a given  $k$ -mer and the other pole represents its reverse-complement. The pole assignment is arbitrary, but once assigned, it is consistent for

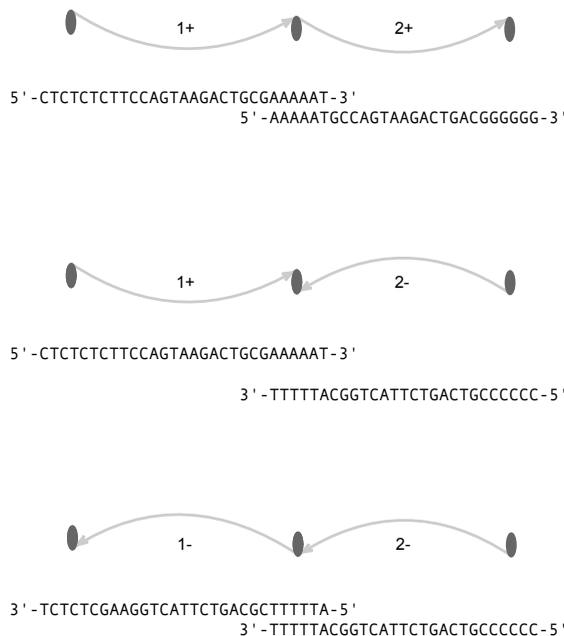


Fig. 4. Example of vertex polarity using an overlap size of 6 nt. Both graph and sequence representations are shown. When a contig edge is reverse-complemented, its orientation, labelled strand, and vertex pole connections are inverted.

all edges. Figure 4 illustrates this encoding. The two example contigs, “1+” and “2+”, share an “AAAAAT” sequence and therefore connect to the same vertex pole (upper panel). When contig “2+” is reverse complemented into “2-” (middle panel), it now has the reverse orientation with the complementary “TTTTA” sequence at its terminal end and therefore connects to the opposite vertex pole. If contig “1+” is subsequently reverse complemented into “1-”, it too now terminates in “TTTTA” and both edges connect to the lower vertex pole (lower panel).

We mapped this reverse-complementation to a simple interaction to enable the user to intuitively manipulate contig orientation. Double clicking a contig edge reverses its direction, switches the polarity of its vertex connections, and changes the sign of its label. This operation, together with being able to freely move vertices, also helps users to clearly see the polarity of an edge connection which can sometimes be obscured in the automated force-directed layout.

This representation maintains the overall ease of interpreting sequence overlaps, however, it does introduce a subtlety with regards to possible paths. Two edges form a valid path through a vertex only if they share the same pole when oriented in the same direction. For example, Figure 4 shows valid paths from contigs “1+” to “2+” and similarly from contigs “2-” to “1-”. We found this rule to be intuitive to biologists as it captures a fundamental property of how DNA sequence is read.

### 4.3 Contig Length

Many analysis tasks require information about contig length. Typically, only low-resolution judgements are needed such as classifying a contig as relatively long or short. One option for representing contig length is to scale the edge lengths corresponding to their sequence lengths, however there are several draw-backs to this approach. First, the layout problem may become intractable as a 2D space may not be able to satisfy all of the distance constraints. Second, even if a layout was resolved, very short adjacent contigs would pack together

and become hard to distinguish. Many assembly ambiguities result from short contigs, so it is essential that the visualization display them clearly.

A wave representation for the contig edges solves these problems. Each oscillation corresponds to a fixed number of nucleotides as defined by the user. Long contigs produce short wavelengths while short contigs result in longer wavelengths (Figure 5). Simple arcs represent contig sequences with below threshold lengths. To ensure that the connections between edges and vertices remain clearly visible and are not overpowered by the wave, we adjust the amplitude along the sequence length setting the minimum amplitude at the connection points. We also use a darker color to distinguish vertices from edges.

Initially, we placed the maximum amplitude at the edge midpoint. In cases of very small wavelengths, we noticed that the edge arrowhead was no longer readily visible, as it tended to blend with the edge’s solid shape. An interpolated color gradient, such as from green to red, has been used to indicate edge direction [12], however in our case we wished to reserve color as much as possible to encode meta-data (see Section 4.6). Other techniques for drawing directed edges include half-lines where only the first half of an edge is drawn [2] or using curvature to indicate direction [7], but neither method is compatible with our use of polar vertices or edge length. We decided to encode contig orientation into the shape itself by forcing the maximum amplitude to occur closer to the contig start. The resulting leaf-like shapes are asymmetric across an edge’s length and thus capture the orientation of the contig sequence. The effectiveness of varying the width of an edge along its length (i.e. wide at the start and narrow at the end) was demonstrated in recent user studies of visualizing directed edges in graphs [13].

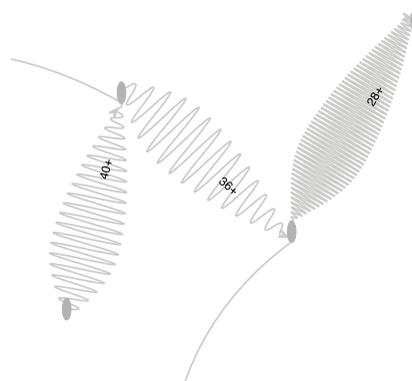


Fig. 5. Contigs of different sizes appear as distinct shapes. In this graph, one oscillation corresponds to 100 nt such that smaller contigs appear as waves (e.g. contig “36+” = 3,000 nt) and larger contigs become solid shapes (e.g. contig “28+” = 11,700 nt).

### 4.4 Contig Coverage

Coverage serves as a valuable metric in many types of analysis. For example, spurious sequencing errors give rise to very low coverage contigs. In contrast, long repetitive elements produce indistinguishable reads along their length which collapse into very high coverage contigs during the assembly process. Extreme coverage values are good indicators of errors or ambiguities and should be easily accessible within an analysis tool.

Color lightness is a natural choice for ranking coverage values such that densely covered contigs appear dark and poorly covered contigs look pale. However, our contig edges already range in their line density due to their different lengths, so judging a gradation of lightness on such variable shapes would be nearly impossible. We also want comparisons between coverage values to be robust independent of color choice. Given that perceived lightness varies with hue, this is not the strongest choice.

We employ line thickness to indicate coverage as it provides an intuitive mapping (more reads piled up results in a thicker line). Four distinct line thicknesses encode four coverage ranges derived using the coverage mean and standard deviation. There is however a trade-off with contig length, as very long contigs may appear as solid shapes and thus the line thickness is obscured. Given that the wavelength along edges is adjustable, we feel that this is an adequate compromise. A user can tune the oscillation frequency such that edge thickness can be made visible for the contig size range of interest. The resulting effect is that contig length and coverage resolve together by permuting a single parameter.

#### 4.5 Read Pair Information

The ABYSS assembly algorithm aligns read pairs to contigs and records this mapping information together with a distance estimate and the orientation for the connection (Figure 6a). The most natural representation for these linking read pairs in the graph consists of a directed edge with a length corresponding to the distance estimate. We use the same length encoding as for contigs, but distinguish linking edges from contig edges through color and the use of a dashed line to give a sense of their virtual distance (these linking edges do not correspond to observed sequences, but rather to inferred inter-sequence distances).

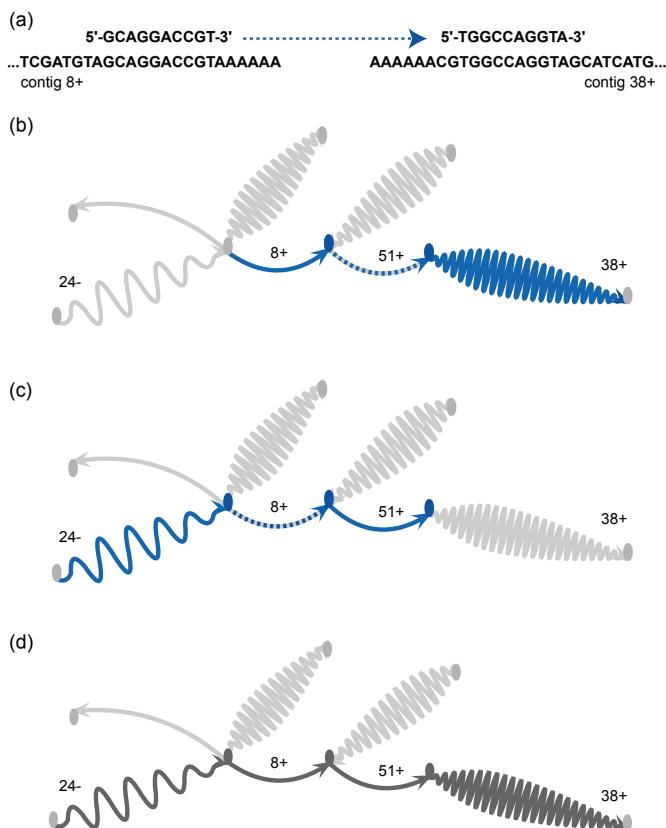


Fig. 6. (a) A read pair (short black sequences) connects contigs “8+” and “38+” (longer blue sequences) and the orientation is captured by a link edge (dashed blue line). (b) In the graph representation, the two contigs involved in the link are colored blue (“8+” and “38+”) and are connected by a dashed blue line with the correct orientation and vertex polarities. (c) A similar link occurs between contigs “24-” and “51+”. (d) The inferred path through these contigs based on the unambiguous read pair information in (b) and (c) is highlighted in dark gray.

One consideration is how to represent the terminal contigs in which the paired reads align. Initially, we simply colored the vertices connected by the new linking edge. This proved problematic because

many contig edges may connect to the same vertex and the contigs containing the read pair remain ambiguous. Instead, we now color the corresponding contig edges to match the linking edge (Figure 6b). If many linking edges for different contigs are displayed simultaneously, the pairing may be difficult to resolve. However we feel this is a reasonable limitation as the linking edges are by default not shown and are only displayed for the selected contig edge. The user can thus easily explore the relations interactively and can control how much information is viewed at one time.

Read pair information is also used by ABYSS to piece together paths through the graph. The presence of a read pair provides additional evidence that two contigs indeed connect and are not adjacent in the graph as a result of a spurious sequence overlap. For example, Figure 6b shows evidence that contig “8+” is followed by contig “38+”. Similarly, Figure 6c supports a connection between contigs “24-” and “51+”. ABYSS will use these unambiguous links to infer a path from contig “24-” to “8+” to “51+” to “38+”. We highlight such paths using a user defined color (dark gray in Figure 6d). Because path color can obscure the coloring resulting from link edges, we encourage users to only view one of these information types at a time in the interactive ABYSS-Explorer.

#### 4.6 Displaying Meta-Data

Color offers an effective way to encode nominal data [20], therefore throughout our design phase, we reserved color as much as possible for displaying meta-data such as categorical annotations. This provides a clean separation between the structural features of an assembly encoded in the forms of the graph’s edges and vertices, and the external meta-data not used in the assembly process. Such separation provides a flexible way to explore diverse meta-data within the structural context of an assembly. Section 6.2 describes how we exploited this encoding in studying large-scale genomic rearrangement events in human cancer.

### 5 IMPLEMENTATION OF ABYSS-EXPLORER

Our assembly visualization tool, ABYSS-Explorer, was designed with our in-house assembly algorithm (ABYSS) in mind, however the details of our implementation and flexibility of our data encoding enable more general use.

ABYSS-Explorer is an interactive system implemented in Java 1.5.0 using the Java Universal Network/Graph Framework, JUNG [16]. This provides us with a well-developed framework that could be easily extended. We use the JUNG implementation of the Kamada-Kawai algorithm for vertex layout [17] and obtained a satisfactory response time on a desktop PC using graphs containing a few hundred vertices. Contig length is only used to render the contig edges and does not influence the graph layout. ABYSS-Explorer runs on DOT formatted input files, consistent with widely used graph visualization software like Graphviz [10]. All input data, such as the contig adjacency information and contig lengths, are stored as DOT files and are parsed with a general DOT parser written using the ANTLR framework [1]. ColorBrewer [11] guided our choice of colors throughout.

### 6 APPLICATIONS

We applied ABYSS-Explorer to a number of ongoing analysis projects at the BC Cancer Agency Genome Sciences Centre. The first tackles the challenges of identifying and resolving structural features within an assembly, and the second explores the flexibility of our tool in integrating meta-data into the assembly context. We conclude with a brief discussion of new application ideas inspired by our initial user experiences.

#### 6.1 Visualizing Assembly Structure

One of the primary motivations for developing ABYSS-Explorer was to facilitate the analysis of assembly structure. Figure 7 shows a visualization of over 200,000 nucleotides of a human genome produced by assembling over nine million sequencing reads. This image gives the user an immediate impression of assembly quality not possible with any existing tool. For example, an ideal assembly would be composed

of a single large contig, however, we can see that there are many small contigs remaining in this assembly and that these short contigs often overlap with several larger ones causing ambiguities. A particularly problematic region composed of many short contigs (top right) can be quickly identified. By clicking on different parts of the graph, the user discovers the paths that are unambiguous based on read pairings, such as the one highlighted in dark gray. Not only does it become clear that there is no path that traverses a substantial portion of the graph, but it is easy to inspect the structural ambiguities where the paths terminate.



Fig. 7. ABYSS-Explorer view of over 200,000 nt of human genome sequence.

ABYSS-Explorer is particularly useful for identifying structural patterns in an assembled sequence. For example, Figure 8 shows a connected set of contigs that form a loop. This cycle is composed of six contigs which together have connections to another six contigs branching off of the loop. Tracing a path such as this one is very difficult using a linear contig display such as Phrapview or Hawkeye, and analysts may resort to drawing graphs by hand to represent the connections. Using our tool, these structures are immediately obvious.

In addition to highlighting structure, ABYSS-Explorer integrates different data types used by analysts. For example, Figure 9a shows a structure with two possible paths: either “24+”, “33-”, “16-” or “24+”, “48+”, “16-”. The encoding of coverage data as edge thickness allows the analyst to quickly recognize that lower contig, “48-”, has lower coverage than any of the other three, which is a common indicator of a sequence error or variant. These connections could simply be a result of spurious overlaps within the assembly, however, an analyst can rule out that possibility by interactively selecting either of the larger flanking contigs and observe that their connection is in fact supported by a spanning read pair highlighted in blue (Figure 9b). Imposing coverage and paired read information onto a structural representation of the contig paths in an interactive fashion greatly facilitates this kind of analysis and is not possible with any other tool.

Next-generation sequencing technologies and complementary assembly algorithms, such as ABYSS, are still in the development stages. ABYSS-Explorer’s ability to display structural information has been valuable to our algorithm designers in allowing them to readily see the consequences of parameter changes. For example, short terminal contigs frequently reflect sequencing errors which tend to be more frequent at read ends. ABYSS has a trimming phase where such contigs below some threshold size are removed. ABYSS-Explorer has proven

to be a useful way to examine which contigs are removed and to judge whether their removal improved the overall assembly. Similarly, glaring assembly errors can be examined to identify weak points in the assembly algorithm and inspire ideas for improvement.

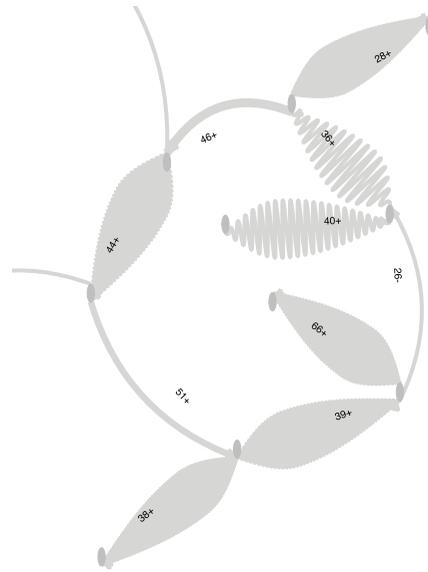


Fig. 8. Detail of an assembled contig cycle.

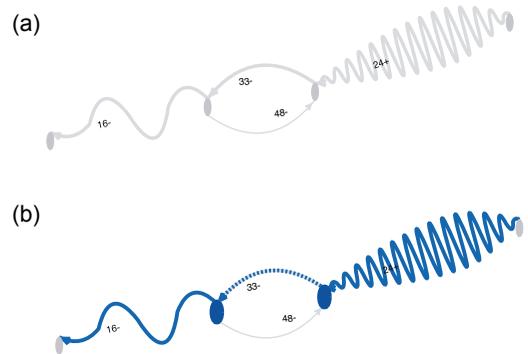


Fig. 9. Detail of an ambiguous path (a) resolved by paired read information (b).

## 6.2 Adding Annotations

While developing ABYSS-Explorer and discussing early prototypes with genome scientists, it became clear that this tool could be valuable for viewing meta-data within the structural context of an assembly. To explore this idea in more detail, we applied our visualization to assembled genome regions from human lymphoma samples identified as containing large-scale genomic rearrangements. The identification of cancer-associated aberrations, particularly for immune cell cancers such as lymphoma, is an ongoing effort. In the final in-house validation stage, contigs of the affected genomic regions assembled by ABYSS are aligned to the reference human genome. Biologists working with these data are interested to know (1) whether the alignment annotations agree with the contig order in the assembly, and (2) whether contigs that map ambiguously using genome alignment can be placed using the assembly.

The upper panels in Figure 10 indicate how we use color to annotate different regions previously discovered to be involved in a genomic inversion event. Our color scheme reflects the order and orientation of sequences in the reference genome (panel a). Blue and

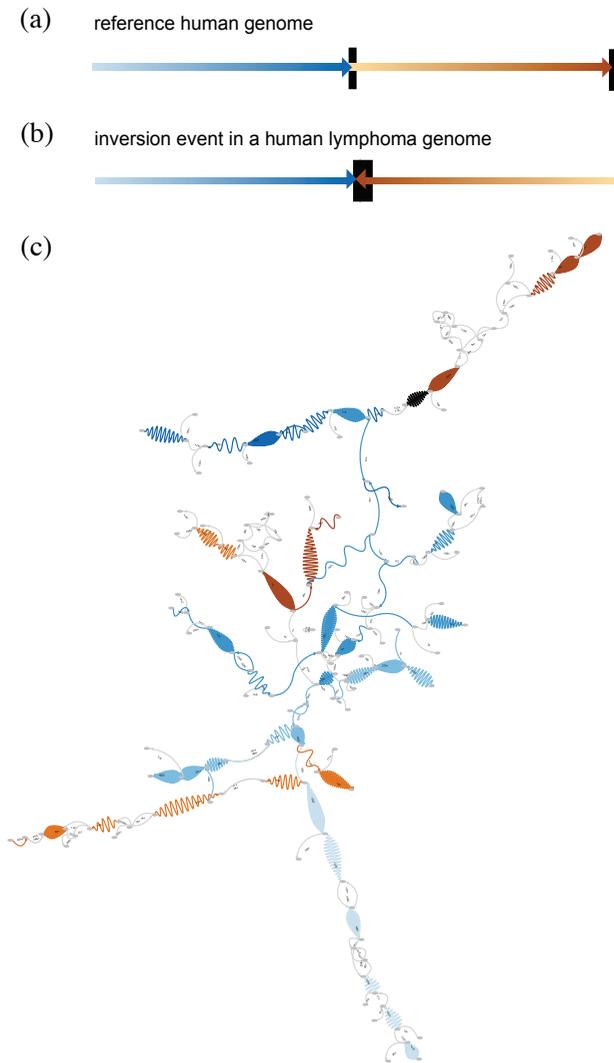


Fig. 10. (a) color scheme annotates roughly 1,000,000 nt of the reference human genome. (b) Same annotations as in (a) indicate a global inversion event in the corresponding region of a lymphoma genome. (c) ABySS-Explorer view of roughly 200,000 nt assembled from the human lymphoma genome region depicted in (b).

orange highlight the two regions of interest and a color gradient indicates distance along the reference sequence. Black boxes mark the end of each region. When we apply these annotations to a lymphoma genome sequence (panel b), we can see that the orange colored region is inverted relative to the reference, creating a novel junction sequence (black box). The graph (panel c) captures an ABySS-Explorer view of the corresponding assembled lymphoma genome region where contigs are colored according to where they align in the reference genome. Contigs with ambiguous genome alignments are left gray.

By imposing these contig alignment annotations onto the assembly structure, we clarify where the assembly order and the alignment order agree. For example, the strings of uninterrupted orange or blue contigs at the bottom of the graph indicate agreement. Inconsistencies are immediately apparent as interconnections between different colored contigs, such as in the center of the graph. It is interesting to note that many of these connection points are made of short contigs, suggesting that repetitive elements or sequence errors are the cause of such ambiguities.

The inversion breakpoint itself is resolved quite clearly, with the

black breakpoint contig flanked by a string of blue contigs on one side and orange contigs on the other (Figure 11). The adjacency of dark orange and blue contigs highlights the nature of the inversion event (compare Figure 10b). Closer inspection of this region reveals a lighter blue contig close to the breakpoint, highlighting an inconsistency between the alignment order (represented by the color gradient) and the assembled order. An analyst must now decide which ordering is correct. Is there an error in the map of the inversion event (Figure 10b) which was deduced using other experimental methods, or is there a mis-assembly? Interactive exploration of the paired read information (Figure 12) reveals that the large suspect contig, now colored dark gray, connects only to smaller contigs mapping to regions further from breakpoint. Because the read pair information supports the alignment order, we suspect a spurious mis-assembly. Interactive data exploration greatly facilitates this kind of analysis.

The detailed view of the breakpoint region (Figure 11) also offers a guide as to where small ambiguously aligned contigs (gray) should be placed. Again, paired read information can be used to make judgements about which contigs belong and which ones may be spurious alignments. Our visualization replaces a time-consuming procedure of inspecting the connectivity between contig pairs in a linear layout viewer such as Consed [9].

Overall, the flexibility of our system to incorporate meta-data such as annotations is very useful for analysts who want to extract biologically meaningful components of an assembly. Given the rapid rate at which the raw data can be generated and assembled and the comparatively slow pace at which the final ambiguities are manually resolved, our visualization enables biologist to make better use of assembly data as it becomes available.

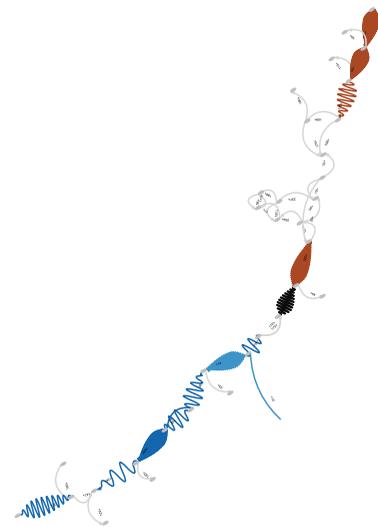


Fig. 11. Detail of the inversion breakpoint in a human lymphoma genome.

### 6.3 Generality of our Visual Encoding

Our initial applications of ABySS-Explorer sparked several ideas for future use, some of which differ significantly from our initial target problem. For example, sequencing technology is not limited to genomes. The genome contains many regions called genes that are transcribed to produce unstable RNA intermediates called transcripts. Many of these transcripts translate into proteins which have a vast array of functions in cells. The collection of RNA transcripts has been coined the transcriptome and can be sequenced in a high-throughput fashion just like a genome.

We are currently experimenting with using the ABySS assembly

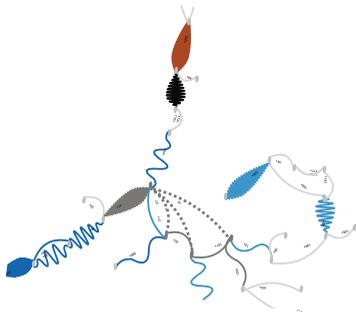


Fig. 12. Paired read information supports the annotated contig ordering.

algorithm to assemble human transcriptomes. Because we sequence thousands of transcripts instead of a handful of chromosomes, the resulting assembly is highly fragmented. Features such as coverage and assembly structure take on new meaning in this context. For example, sequence coverage is highly informative about expression level, and assembly structure can reveal alternatively spliced transcripts. Although this work is ongoing, the initial ease with which we have been able to apply ABySS-Explorer to this new problem domain exemplifies the generality of our visual encoding of sequence data.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper we explored the use of a graph as a novel encoding for genomic sequence data and experimented with how to map properties of DNA onto the forms of graph vertices and edges. Through two case studies, we demonstrated the utility of our sequence representation in tackling distinct analysis problems, both of which previously required manual data integration due to a lack of appropriate tools. Reserving color for meta-data display results in great flexibility to explore the distribution of diverse data types over the assembled sequence. The unified encoding presented here is readily applicable to other biological domains, such as transcriptome analysis.

All of the case studies presented here focus on a sub-region of the human genome constructed from hundreds of contigs which correspond to hundreds of thousands of nucleotides. However, an assembly of a full length human genome corresponds to roughly 3 billion nucleotides and typically assembles into millions of contigs. This scaling problem poses significant challenges in both how to navigate the detailed view and how to provide a meaningful overview. Future work in this direction can build on previous graph visualization research in other application areas while using the novel data encoding presented here.

In addition to the scale problem, there is interest from analysts to enable assembly editing as is available from most popular genome assembly tools. This introduces the notion of graph history and opens up interesting design problems of how to explore changes in graph structure over time. Methods to animate the untangling of a genome assembly over iterations of new data incorporation and manual edits are likely to result in insights into the strengths and weakness of our current genome editing strategies. We believe that graph-based visualizations of assemblies will have even more benefits than those discussed here when applied to histories of changing assemblies, especially when compared to the current text based reporting methods.

## ACKNOWLEDGMENTS

The authors wish to thank Martin Wattenberg for his support and creativity, in particular, for his suggestion to encode sequence length in a wave. We also thank Peter Gorniak for his technical guidance throughout the development of this project, and critical reading and comments on the text. Finally, we wish to acknowledge Jenny Qian, Karen Mungall, and Greg Taylor for helpful discussions and feedback, and the anonymous referees for their helpful comments on this paper.

## REFERENCES

- [1] ANTLR. Another tool for language recognition. [url](http://www.antlr.org), <http://www.antlr.org>.
- [2] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1:16–28, 1995.
- [3] J. K. Bonfield, K. f Smith, and R. Staden. A new dna sequence assembly program. *Nucleic Acids Res*, 23(24):4992–9, Dec 1995.
- [4] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe. Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Res*, 18(5):810–20, May 2008.
- [5] M. J. Chaisson, D. Brinza, and P. A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res*, 19(2):336–46, Feb 2009.
- [6] M. J. Chaisson and P. A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res*, 18(2):324–30, Feb 2008.
- [7] J. D. Fekete, D. Wang, A. Aris, and C. Plaisant. Overlaying graph links on treemaps. *IEEE Symposium on Information Visualization (Proceedings of Infovis 2003)*, Poster Compendium:82–83, 2003.
- [8] B. Fry. *Visualizing Data*. O'Reilly Media Inc., 2008.
- [9] D. Gordon, C. Abajian, and P. Green. Consed: a graphical tool for sequence finishing. *Genome Res*, 8(3):195–202, Mar 1998.
- [10] Graphviz. Graph visualization software. [url](http://www.graphviz.org), <http://www.graphviz.org>.
- [11] M. Harrower and C. Brewer. Colorbrewer.org: An online tool for selecting color schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.
- [12] D. Holten. Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, (Proceedings of InfoVis 2006), 12(5):741 – 748, 2006.
- [13] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. *Proceedings of CHI*, pages 2299–2308, 2009.
- [14] W. Huang and G. Marth. Eagleview: a genome assembly viewer for next-generation sequencing technologies. *Genome Res*, 18(9):1538–43, Sep 2008.
- [15] T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, T. Down, R. Durbin, E. Eyraas, J. Gilbert, M. Hammond, L. Huminiecki, A. Kasprzyk, H. Lehvaslaiho, P. Lijnzaad, C. Melsopp, E. Mongin, R. Pettett, M. Pocock, S. Potter, A. Rust, E. Schmidt, S. Searle, G. Slater, J. Smith, W. Spooner, A. Stabenau, J. Stalker, E. Stupka, A. Ureta-Vidal, I. Vastrik, and M. Clamp. The ensemble genome database project. *Nucleic Acids Research*, 30(1):38–41, Jan 2002.
- [16] JUNG. Java universal network/graph framework. [url](http://jung.sourceforge.net), <http://jung.sourceforge.net>.
- [17] T. Kamada and S. Kawai. An algorithm for drawing general indirect graphs. *Information Processing Letters*, 31(1):7–15, 1989.
- [18] W. Kent, C. Sugnet, T. Furey, K. Roskin, T. P. TH, A. Zahler, and D. Haussler. The human genome browser at ucsc. *Genome Research*, 12(6):996–1006, June 2002.
- [19] P.-G. Kim, H.-G. Cho, and K. Park. A scaffold analysis tool using mate-pair information in genome sequencing. *J Biomed Biotechnol*, 2008:675741, Jan 2008.
- [20] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.
- [21] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proc Natl Acad Sci USA*, 98(17):9748–53, Aug 2001.
- [22] M. C. Schatz, A. M. Phillippy, B. Shneiderman, and S. L. Salzberg. Hawkeye: an interactive visual analytics tool for genome assemblies. *Genome Biol*, 8(3):R34, Jan 2007.
- [23] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343, 1996.
- [24] J. Simpson, K. Wong, S. Jackman, J. Schein, S. Jones, and I. Birol. Abyss: A parallel assembler for short read sequence data. *Genome Res*, Feb 2009.
- [25] D. R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18(5):821–9, May 2008.