Epiviz: interactive visual analytics for functional genomics data

Florin Chelaru[1], Llewellyn Smith[1,2,3], Naomi Goldstein[1,4], Héctor Corrada Bravo[1]

[1]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, Maryland, USA

[2]Department of Mathematics, Williams College, Williamstown, Massachusetts, USA

[3]Department of Computer Science, Williams College, Williamstown, Massachusetts, USA

[4]Dept. of Mechanical Engineering and Materials Science, Washington University in St. Louis, St. Louis, Missouri, USA

Corresponding author

Correspondence should be addressed to H.C.B. (hcorrada@umiacs.umd.edu)

Supplementary Note

Detailed architecture

The architecture of Epiviz is organized in three tiers, namely user-interface, visualization and data management. The independence of these layers in key to Epiviz' ability to represent the same data using different visualizations, and reusing the same visualization for different types of data, integrated from different sources. The communication between tiers is managed by a controller which listens to events from each of the three tiers and propagates events as required (Sup Figure 1). URLs for plugins and extensions can be declared dynamically on the request URL to Epiviz or by specifying settings in a JavaScript file that can also be specified dynamically on the Epiviz request URL.

The user-interface tier contains the visual layout and standard genome browsing controls (navigation, gene search, etc.) as well as a framework for the other components of Epiviz. It is designed to adapt to the plugins in the visualizations and data management tiers, by displaying UI items (e.g., menu entries in measurement selection dialogs) dynamically for plugged-in data or visualization. Thus, a user can access Epiviz online, specifying custom visualizations, without downloading a copy of the software on their local machine. In fact, as mentioned in the Results section of the main manuscript, if the custom visualizations are hosted online (for example, on GitHub's Gist service), they can easily be shared among users without the modification of the Epiviz base code. This design encourages the scientific community to actively contribute to Epiviz without the bottleneck of a production pipeline.

The visualizations tier implements a framework that processes all the available visualizations, whether they are predefined on the hosted server, or specified at runtime. The visualizations themselves are created using the d3.js library, which allows the display of data using scalable vector graphics (SVG), a flexible format that can be easily saved or converted to static images. Again, JavaScript files defining new visualizations can be added dynamically by specifying URLs in settings.

The data management tier serves data to the user-interface tier from multiple data providers. Epiviz has two predefined types of data providers: a) one that acts as a proxy to a PHP-MySQL web server hosted at the University of Maryland; and b) one that uses a WebSocket API to connect to programming environment that supports the protocol. The latter is used by the Epivizr Bioconductor package to establish interactive sessions between R and Epiviz. When a user takes an action in Epiviz, like adding a new chart or navigating to a new genomic location, asynchronous data requests are sent through the data management tier to these data providers. In turn, through these data providers, users can dynamically send commands to the Epiviz UI, adding new measurements and charts or navigating. Data from all these providers is integrated at the data management level so that the UI is only aware of a generic data source. To improve user experience, we have implemented a client-side predictive caching strategy in the data management tier to accelerate system response to user-initiated data requests. The caching mechanism, used regardless of where the data is served from, is fairly simple in concept, requesting data in advance for predicted user actions. The requests from the UI first stop at the cache, which, if the requested data is already available, serves it back without sending further requests to the data providers. If all or part of the requested data is not available in the cache, the cache sends three requests to the corresponding data provider: one for the requested data, and two for an equal amount of data for the genomic regions immediately before and after the requested region. These requests preempt requests that would be made by navigating to neighboring genomic regions, and when zooming out of the current genomic region. The resulting user experience is that the user only has to wait once for the initial request to be fulfilled, all other requests being done in advance, taking advantage of user interaction idle periods. Our performance analysis shows substantial performance improvements for adding new charts and navigation when using the cache (Sup. Figure 2).

Functional genomics data can be referenced in two distinct domains: by genomic location, and by genomic feature. In the former, data is organized and referenced by location: for example, regions of interest (ROI) defined by start and end genomic locations, or continuous measurements like DNAm obtained at base-pair resolution. Otherwise, data is organized and referenced by feature: for example, expression for a gene, transcript or exon. Organizing and referencing data in multiple domains requires support for different types of data visualizations. This is a central aspect of the Epiviz design. Data in the location domain is displayed in tracks where visualization appropriate for spatial data is required while data in the feature domain is displayed in charts where other visualization approaches, like scatterplots, heatmaps, etc., are more appropriate.

The design used in Epiviz allows logical separation of datatypes (measurements, such as genomic ROI, continuous measurements along the genome, feature-based measurements such as exon, transcript or gene-level expression), from their visualization. These are represented as distinct object classes in the JavaScript Epiviz framework, allowing for an extension mechanism whereupon new datatypes or new visualizations can be implemented independently. This logical separation stresses the fact that multiple visualizations of the same data can be created simultaneously in Epiviz.

To further support interactivity in Epiviz, we added support for a simple expression language that users can use to define new data measurements based on measurements loaded to the data management tier. We use the JavaScript Expression Evaluator (http://silentmatt.com/javascript-expression-evaluator/) for this purpose. The expression syntax definition can be obtained there. This feature allows for iterative visual analytics, which are imperative in the exploration of functional genomics data. Epiviz also supports other standard genome browser functionality like search by gene symbol or Affymetrix probe id. Epiviz provides detailed information on data measurements and data sources through information tooltips on demand, activated by hovering over objects in visualizations.

Epiviz allows users to save and share workspaces that store genomic location information along with the data measurements and UI elements displayed in a session. This allows users to seamlessly share a visual analysis for presentation or as part of a collaborative project. Workspace links are persistent allowing analyses on Epiviz to be referenced in publications. For instance, Figure 1 can be accessed at http://epiviz.cbcb.umd.edu/?ws=cDx4eNK96Ws. Persistent workspaces for sessions hosted at http://epiviz.cbcb.umd.edu are stored in the data server hosted at the University of Maryland. However, persistent workspace storage management is part of our data provider API (see below) and source code for this type of server is publicly available on the Epiviz project page and can be installed on a standard PHP-MySQL system to provide the same functionality if users desired to keep their workspaces private.

Data providers as plug-ins: Currently, Epiviz has two predefined types of data providers described earlier: a proxy to the PHP web server (which it communicates with through GET or POST requests), and a proxy to a WebSocket API which allows connections to any programming environment that supports this protocol – like R through Bioconductor. In order to accommodate the needs of different types of users, we have exposed a plug-in API which users can use to add new data providers that implement a particular interface on-the-fly. Epiviz supports simultaneous connections to multiple instances of these data providers (for example, multiple web servers or WebSocket sessions combined), thus allowing an easy integration of multiple data sources at the same time and a high level of collaborative data analysis. Management of persistent workspace storage is also part of the Data Provider API. Therefore, users can indicate through the settings file the URL of the Data Provider that resolves workspace identifiers per individual session.

New data management technologies to support interactive, data-rich web applications like Epiviz is a very active area of development in the data analytics community. As a result developing the Data Provider used in *Epivizr* we added support for repeated querying of Bioconductor data by implementing IntervalForests, a collection of interval trees (one for each sequence in an assembly) for efficient querying of data by genomic location, which is the prevalent query operation in Epiviz. This extension is now part of the Bioconductor code base and available for general efficient overlap operations over genomic intervals as part of the Bioconductor infrastructure. Data on the proxy server hosted at the University of Maryland is also indexed by genomic location to improve querying performance. As new Data Providers are created and plugged-in, specific efficient indexing techniques for those data sources can be implemented as well.

Unified data types: The Data Provider API defines a JSON format for data exchanged between data sources and the Epiviz application. All transferred data, independent of its source, is standardized into one generic format, which is used by all visualizations. The design of this generic format is based on the SummarizedExperiment design in Bioconductor*,* which draws from the “three-table” design for genomic data: each data point is part of an assay*,* and it measures a particular feature from a specific sample, each of which is annotated by data themselves. Specifically, features are genomic coordinates, regions or set of regions. The data exchange format in Epiviz follows this design where data rows determine genomic coordinates and other annotations (exon or gene id for example), and values in these rows are measurements to display from multiple samples, each with its own annotation. We find that this design is capable of modeling the vast majority of data present in functional genomics experiments. The full description of the data format exchange is available in the Epiviz online documentation. Each Data Provider is responsible for parsing whatever data is serving into this unified type to transfer data to the UI. Therefore data parsing for integration is moved from the UI to the data providers and libraries that implement connections to the Epiviz API.

The unified data types used in Epiviz include identifiers for datasources, and datasourceGroups which declare metadata for each set of observations that are used as keys to establish data relationships used in, e.g., the brushing feature described previously. Therefore all data sources from the same group are assumed to have the same metadata available which is then used as a common key. Notice that this allows these keys to be defined dynamically. In the absence of keys we use genomic location overlap to establish these relationships as well.

Optimizations for multiple resolutions: The predefined visualizations in Epiviz support viewing high and low resolution data side by side with full brushing support. Low resolution views group data objects together into one visual object, in order to avoid cluttering the screen with too many vector objects, thus increasing the performance and improving the user experience of Epiviz. For example, in line plots, we use a parameter indicating the maximum number of points that may appear on screen. If the number of data points in a region is greater than this parameter, we sample the required number uniformly at random and only include that sample in the line plot. Each of the standard visualizations are similarly parameterized to allow the user to balance between loading/rendering speed and level of visualization detail. These parameters can be specified in settings files. We report below on the performance behavior while varying these parameters for the four predefined Epiviz visualizations: scatter plot, heatmap, blocks track and line track.

Performance benchmarks

Epiviz features two types of speed optimizations that facilitate quick display of visualizations: predictive caching, and chart-specific grouping of data elements into visual elements. The first one attempts to reduce the data retrieval and processing time on a new user action; the second attempts to reduce visualization rendering time. Together, they minimize the time taken between the time a user executes an action on the screen and when they are able to interact with a view of the requested data.

We implemented a suite of performance tests, measuring total time taken by the two most common user actions – ‘add chart’ and ‘navigate’, for our predefined visualizations – Scatter Plot, Heatmap, Blocks Track and Line Track. For both operations, we compared performance with and without the predictive cache. Our results reveal that using the cache speeds up execution by more than 25 times for all visualizations (Sup Figure 2).

We also measured the relationship between the number of data objects available to draw, and the number of objects drawn, showing how that affects the total draw time of charts (Sup Figures 3 and 4). For each chart, we varied specific parameters that determine the number of drawn objects:

- Scatter Plot: ‘Circle Ratio’. This parameter sets the radius of the drawn data points, in relation to the minimum of the chart height and width. The visualization is programmed to split the chart in a grid of squares of width equal to this parameter, and draw at most one circle in one cell of the grid. Thus, if multiple data points overlap the same grid cell, only one is drawn. Object transparency is used for underlining locations with higher density of points in one location.
- Blocks Track: ‘Min Block Distance’. This parameter sets the minimum distance in screen pixels between two blocks before they are merged into one. If the parameter has a large value, more blocks will be merged together, causing less objects to be drawn on the screen. Setting this value to 0 disables the parameter.
- Heatmap: ‘Max Columns’. This parameter sets the maximum number of columns to be drawn by the heat map before multiple columns are averaged into one. When the number of available columns is greater than this number, they are split into a number of groups equal to this number and only one column is drawn for each group.
- Line Track: ‘Max Points’. This parameter sets the maximum number of points to be drawn by the line track. If the number of available data points is greater than this number, the points are sampled uniformly across the requested genomic range, so their number matches this value.

Tuning these parameters as the requested data ranges increase offers users the power of adjusting their user experience and also a straightforward way of summarizing data. Our results show how the performance of Epiviz dramatically increases with appropriate values for these parameters.