OnSet: A Visualization Technique for Large-scale Binary Set Data

Ramik Sadana, Timothy Major, Alistair Dove and John Stasko

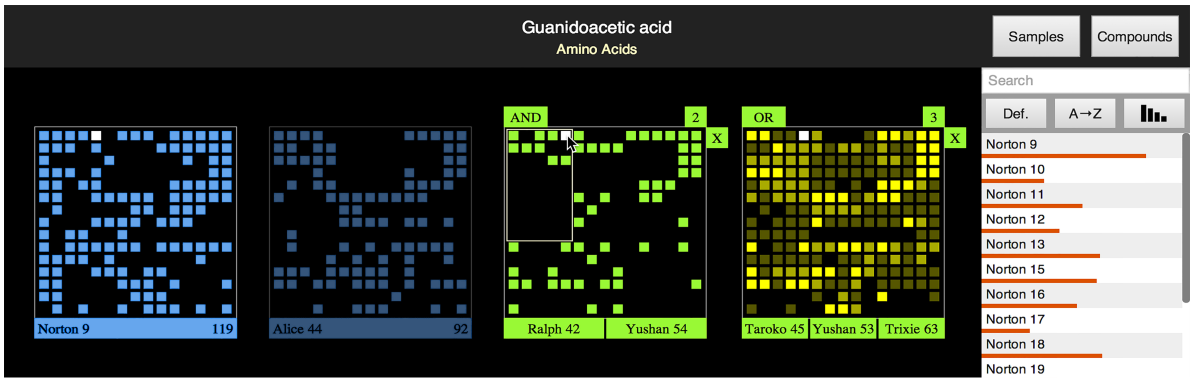

Fig. 1. User interface of OnSet. The list of sets is on the right (the orange bar indicates the number of elements in the set). Each set is represented as a rectangular region with the set elements being smaller interior rectangles. This view shows two individual sets (one is dimmed) on the left and two compositions of individual sets on the right.

Abstract—Visualizing sets to reveal relationships between constituent elements is a complex representational problem. Recent research presents several automated placement and grouping techniques to highlight connections between set elements. However, these techniques do not scale well for sets with cardinality greater than one hundred elements. We present OnSet, an interactive, scalable visualization technique for representing large-scale binary set data. The visualization technique defines a single, combined domain of elements for all sets, and models each set by the elements that it both contains and does not contain. OnSet employs direct manipulation interaction and visual highlighting to support easy identification of commonalities and differences as well as membership patterns across different sets of elements. We present case studies to illustrate how the technique can be successfully applied across different domains such as bio-chemical metabolomics and task and event scheduling.

Index Terms—Set visualization, information visualization, direct manipulation, Euler diagrams, interaction, logical operations.

1 Introduction

A set of elements is one of the simplest organizational structures for data. The presence of multiple sets and large numbers of set elements adds to the complexity. When particular elements are contained in multiple sets, understanding the commonalities, differences, and relationships among the sets becomes more challenging.

Consider a situation where samples of a substance are taken from different locations at different points in time. The substance is made up of a set of constituent components, but only a subset of all possible components is ever present in a sample, and those subsets vary widely in make-up and size. An investigator who has gathered a collection of samples may seek to identify common component patterns across the samples, may look for the presence or absence of key components, and may need to understand how the subsets of components are distributed across the samples.

Membership, difference, intersections, and other metrics on the samples can be characterized formally and calculated precisely, but an investigator may not be seeking the answers to specific questions. Instead, they may be exploring the collection of samples to learn about its characteristics and to discover new information.

- *Ramik Sadana is with Georgia Tech. E-mail: ramik@gatech.edu.*
- *Tim Major is with Georgia Tech. E-mail: tmajor3@gatech.edu.*
- *Alistair Dove is with the Georgia Aquarium. E-mail: adove@georgiaaquarium.org.*
- *John Stasko is with Georgia Tech. E-mail: stasko@cc.gatech.edu.*

Manuscript received 31 March 2014; accepted 1 August 2014; posted online 13 October 2014; mailed on 4 October 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

The visualization of sets and their elements has been a helpful tool for assisting situations such as this. A Venn diagram [12] is a valuable representation tool for showing shared and unique elements among two or three sets. An Euler diagram [9] generalizes the notion of a Venn diagram and can be used to represent larger numbers of sets. As the number of elements and sets rises, however, the complexity of an Euler diagram rises as well. It can be difficult to create a comprehensible, aesthetically pleasing view of set-element relations even for very modest numbers of sets and elements.

The visualization research community recently has developed a number of techniques to represent flexible collections of set elements [1, 7, 10, 14, 17, 20]. Increasing quantities, both of elements and sets, remains a challenge though, and these techniques quickly become complex and difficult to visually interpret and understand as the sets contain larger numbers of elements. More recent techniques explicitly address scalability [2] but they sometimes come at the cost of showing individual elements and the views become more difficult to interpret.

We have created a new visualization technique to portray collections of sets whose cardinality (number of elements) may be in the hundreds. Our technique is called OnSet and it combines a grid style representation with a suite of interactive operations that help to compare and contrast different sets. The technique differs from most other set visualizations in that it does not follow the constraint that an element in the collection can have only one visual representation or glyph. Instead, OnSet uses a consistent position within all set representations to indicate a particular element. Clever application of highlighting and brushing then assists in the identification of commonalities and differences. Additionally, the use of direct manipulation interaction to stack (compose) flexible combinations of sets is fundamental and crucial to the technique.

The primary contribution of this research is the OnSet interactive visualization technique. We begin the article by describing set-relevant concepts and defining the terminology we use throughout. Next, we review related work and describe the types of analytic inquiries that are common when working with sets. After explaining the OnSet technique in detail, we conclude by describing two applications of it. The first comes from the domain of bio-chemical metabolomics and the second involves calendar scheduling. We implemented visualization applications to embody the OnSet technique in both domains.

2 Background

A set is an unordered collection of items (elements) with no repeated values. Two well-known methods for visualizing sets, Venn and Euler diagrams, have been widely used since the 18th century to show properties and relationships within collections of sets. Using these techniques, it is possible to view the results of several important operations. For example, the intersection of two sets, A and B, is defined as the set of elements that are found in both A and B, or in other words, it answers the question "What are the common elements of these two sets?" In an Euler diagram, the intersection can be found by locating the items within nested boundaries. Other operations include union ("What are all of the elements in any of these sets?"), and complement or subtraction ("What elements are in this set but not in the other?").

An interest in visualizing set data has continued to the present day. Below we review a number of set visualization techniques and systems. Throughout all these projects, a number of challenges and limitations persist. One concerns the size of sets being displayed. Because of the representation methods chosen and the need to show set intersections, displaying sets with large cardinality often results in highly complex views that are difficult to understand. Even for sets with fewer elements, complex views can result if many sets share a high number of common elements. In such situations, set visualization techniques often employ visual properties such as color and opacity to depict set membership. However, identifying the correct set or sets that an element resides in becomes fairly difficult. Finally, for complex sets, creating these visual representations can be a computationally hard process.

Various datasets exist that encounter these limitations. Consider the data depicting the code repositories of a large organization, such as found on Github. Each project can be thought of as one set, with the elements of the set being the libraries that the project requires. A large-sized project could link to anywhere from 20 to 100 different libraries. A company could have five or more of these projects, where some or all of the projects share many of the libraries with each other. The visualization would then attempt to portray all of the distinct libraries in a single view, with boundaries drawn to show set membership among the different projects. Since the number of libraries could easily be more than 200, each would be represented as a very small glyph. In addition, the boundaries of the project sets form a complex and convoluted web. Interaction becomes crucial, as there is not much the user is able to understand about the data right at the outset.

2.1 Related Work

Techniques for visualizing sets have a long history. For a comprehensive review of set visualization techniques, see [3]. Below we review some of the techniques most germane to our research.

Venn diagrams are among the most well-known. Conceived in the 1880s by John Venn, they have been adopted as a common means of teaching set theory and logical relations in classrooms [12]. A Venn diagram consists of a group of closed curves, most often circles, with overlapping boundaries used to display all logical relations for the sets represented by the closed regions. A Venn diagram is a special form of Euler diagram, a technique for visualizing sets that dates back to the 1700s and earlier [16]. An Euler diagram also uses enclosed contours to denote set boundaries. However, unlike a Venn diagram, an Euler diagram does not need to contain all 2^n possible relations for n sets. These relaxed constraints help reduce the complexity of the visual representation compared to Venn diagrams, but the complexity still rises quickly as the number of elements and sets increases.

Many visualization techniques deal with the contours and layout of Euler diagrams. The BubbleSet technique overlays set boundaries on existing visualizations without interfering with the original layout [7]. The technique draws isocontours around set elements at interactive speeds without using layout techniques to disturb them. Simonetto, Auber, and Archambault [20] present an algorithm that is able to automatically draw an Euler diagram for any input. They use Bézier curves and transparent colored textures in order to make the diagrams more comprehensible. Riche and Dwyer [17] also seek to reduce the visual complexity of Euler diagrams by using hierarchical data structures to construct two possible layouts. They created the Compact Rectangular Euler Diagrams (ComED) technique that uses compact rectangular areas to draw the sets in order to improve readability and the Euler Diagrams with Duplications (DupED) technique that allows the duplication of elements in the drawn sets to remove overlapping set boundaries.

Euler and Venn diagrams are not the only ways to visualize sets, however. The LineSet technique [1] uses colored curves to "connect the dots" between elements of a set. This allows set membership to be overlaid on existing visualizations, like the BubbleSet technique, but with a simpler marking system. The ConSet technique uses a matrix representation to display set data [14]. Columns in the matrix represent individual elements, while sets that contain the elements are represented by the rows. The presence of an element in a set is indicated by filling in the cell corresponding to the element/set pair. In addition to the matrix representation, ConSet also provides an alternative to a Venn diagram called a Fan diagram. A Fan diagram displays set data by partitioning areas of a circle much like a pie chart. Each area represents a set. However, the areas are allowed to overlap to show elements that are shared between sets. This representation has the advantage of directly mapping the number of elements in each set to the size of each partitioned area.

Several items of previous work focus on features leveraged by the OnSet technique described in this paper. The Set'o'grams technique [10] relies heavily on brushing to show how elements occur across several sets. In this representation, element groups are distributed across several histogram-like blocks representing different categories/sets. Each element group may belong to multiple categories. As the mouse hovers over an element group, the same group is highlighted in the other blocks where they are found. The Radial Sets visualization [2] seeks to address issues of scale that arise with several of the techniques discussed so far. They extend several ideas, such as frequency aggregation from the Set'o'grams, to produce a tool that can analyse sets with large numbers of elements. However, this technique does not scale well to a large number of sets due to visual complexity.

The PIWI system [22] uses a matrix representation to depict graphs, where each position denotes a vertex. The matrices effectively represent different subsets of the graph, and the user can perform different Boolean operations on the subsets. Flexible direct manipulation of the matrix subsets is not provided, however.

One approach to dealing with issues of scale is to leverage "dense pixel displays" [13]. In such displays, data items or datasets are represented by visual objects or glyphs in which each pixel in the glyph corresponds to a data case or attribute. These visualizations leverage the human visual perception system to quickly spot outliers and trends among data items scaled down to the size of pixels on the screen. They also support the display of many items due to the relatively small size of each item. One example of such a technique is the VaR system [21] that scales data items down to dense pixel displays based on dimensions of the data. These dimension "glyphs" are positioned on a canvas such that similar data cases are near each other. Our OnSet technique takes a similar approach and scales down sets with large cardinalities and places them on a canvas. However, direct manipulation of the representations is fundamental for the exploration of relationships between the pixel displays. In this respect, OnSet incorporates ideas from the Dust & Magnet system [23] that portrays data cases as iron dust and data variables as magnets that attract or repel the dust. In this technique, direct manipulation of the data is vital to exploration and sensemaking.

Many of the above-mentioned techniques can be classified as being element-driven. The focus of the view is on individual elements and an element is present just once in the visualization. A set, then, is defined by a boundary that encompasses one or more of these elements. Repositioning an element with interaction morphs the structure or design of the set. However, a different way to view set data is possible. Taking a set-driven approach makes the basic unit of representation a set. These sets are made up of the elements they *can* contain, rather than what they *do* contain. Because of this focus, an element can be duplicated across the visualization. And although the position of a set can be modified, the underlying structure of the set remains consistent. This consistent structure is a necessary requirement for locating elements in the set, particularly when they are contained by multiple sets. We have adopted this latter methodology for the OnSet technique.

2.2 Analytic Tasks on Sets

A wide variety of analytic inquiries are possible about set collections. Alsallakh et al [3] provide a list of common tasks with set-typed data. The list is divided into three main categories: tasks related to elements (e.g., Find sets containing a specific element), tasks related to sets and set relations (e.g., Identify intersections between *k* sets), and tasks related to element attributes (e.g., Analyze the set memberships for elements having certain attribute values). Beyond these relatively concrete tasks, we envision analysts engaging in more open-ended sensemaking sessions where they explore set collections simply to learn more about the set data and to gather insights about it. In all likelihood, the particular domain of the data will dictate the specific types of questions and tasks that arise.

Our work does not focus on tasks involving set elements with multiple quantitative and categorical attributes. Rather, the elements have a single attribute – such as a unique name or numerical value – that is used to distinguish them from one another and to determine their membership in the sets. We do, however, include the notion of related groups of set elements when those elements can be considered to come from the same category. At the end of the article we address a possible extension to our work that would incorporate additional attributes of the set elements.

3 Design Explorations

At the onset of our research we considered a variety of visual designs to represent large cardinality set data. Our initial explorations were motivated by the need to maximize the amount of data the user could see in an initial view. This aligned with presenting an overview first, per Shneiderman's oft-repeated InfoVis mantra [19]. To pack as much data as possible on the screen, we used a fisheye lens [11] approach. We employed a matrix representation for the data, with each row corresponding to an element in the data and each column corresponding to a set. A particular position (x, y) could take one of two states: present if the set y contained the element x, and absent otherwise. The present and absent states were depicted in blue and black color respectively, while red indicates the currently selected element. Figure 2 presents the resulting visualization, which looks similar to the technique used in the TableLens [15]. However, in our case, each element only presents a boolean state (present or absent) and not a quantitative or categorical value.

In the default state, the height of each row was two pixels, with a gap of two pixels between pairs of rows. Hovering over a row expands that row of pixels and highlights the name of the element. Tracking an element horizontally across a row gives information on how frequently the element occurs in different sets. Similarly, tracking vertically along a column highlights the number of elements a set contains. This value can also be used to sort the sets so that the sets with largest number of elements are placed in column 1 and so on. Finally, we wanted to support the comparison of two different rows. Such a comparison helps in highlighting how often the two elements appear across sets. Since row comparisons can best be performed between adjacent rows, we provided the option to drag a row vertically to place it between any other pair of rows. Alternately, a row could be highlighted or bookmarked by double clicking on the name of the element. This would color the row blue, which makes it easy to spot and zoom to later. Finally, the visualization also supported search. The user could type a search query in the search box at the top to highlight and expand all the rows that contained the search term.

Unfortunately, the design suffered from a few limitations. First, only a limited number of elements could be shown vertically. Since each element took a total of 4 pixels to represent, only about 200 elements could be placed vertically. For any set that contained more than 200 elements, the user would need to scroll, which added substantial complexity to other aspects of the interaction. A similar limitation existed on the number of sets that could be shown, in which case the user would need to resort to a horizontal scroll. Another issue was the comparison of different sets. Although the visualization was fairly useful for comparing elements across different sets, such comparisons were not feasible for the different sets. The visualization also did not easily communicate information about either the total number of elements or the number of sets in the data.

To address these concerns, we designed another version of the visualization, as shown in Figure 3. In this iteration, we primarily wanted to support the easy comparison of sets. We removed the matrix representation and switched the position of the sets from the columns to the rows. Each row now showed the elements contained in a set. The elements wrapped over to multiple rows, which made it easier to immediately identify the sets that contained the most

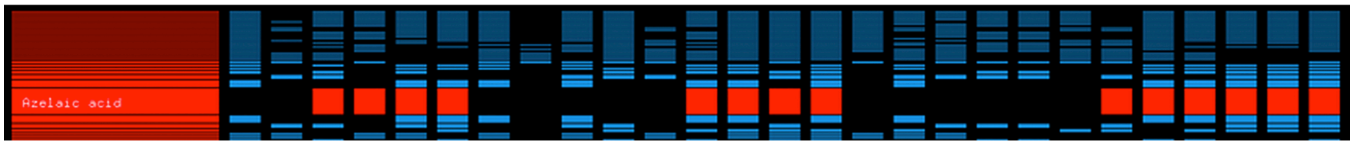

Fig. 2. Initial design using a fisheye approach with elements in rows and sets in columns.

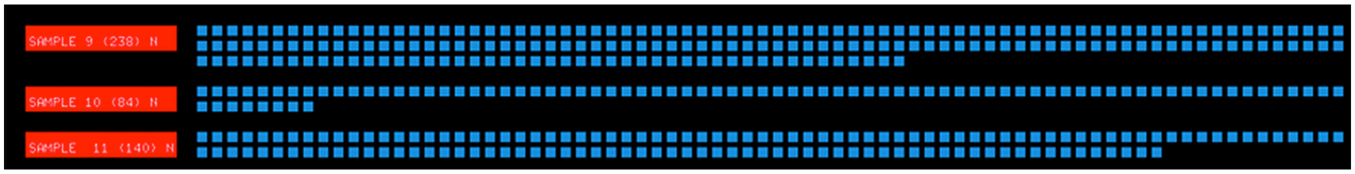

Fig. 3. Second design with sets in rows and elements as blocks pushed left.

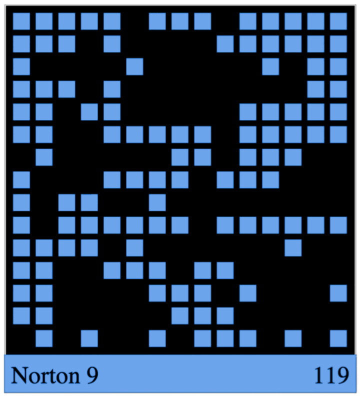

Fig. 4. A PixelLayer representation of set data. The number in the bottom right indicates the number of elements in the set.

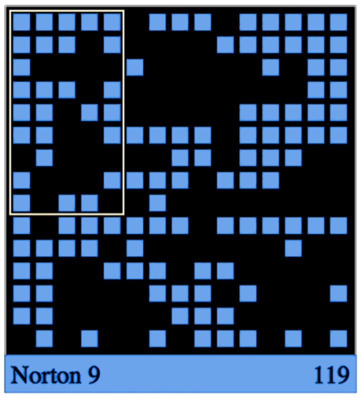

Fig. 5. Hierarchal groups are displayed by an off-white border.

number of elements. Hovering over an element highlighted all other sets that contained it. This design clearly showed the total number of elements present in a set.

One downside of this design was that it was difficult to ascertain if a particular element existed in a set without hovering over each element. Another shortcoming was that this representation only showed a limited number of sets on the screen at any time. Additionally, it did not support easy set comparison due to elements being in different positions in different sets and, thus, being hard to locate. Also, our trial use of the visualization revealed that automatically loading all the sets at start-up was not needed. Instead, sets could be loaded on demand.

We followed this design cue to remove the loading of all sets at the initialization of the interface. In OnSet, the user decides which sets to load onto the canvas for exploration. All comparative analysis such as similarity metrics are calculated only for those sets that are currently on the canvas. Another modification from the original designs was to fix the position of all elements across sets. This characteristic was present in the first, matrix style representation but was removed in the second design. We realized that it was vital for the user to locate an element across all the sets; the presence or absence of a set was one of the chief questions that this visualization helped to answer.

4 OnSet Visualization Technique

After reflecting on the lessons learned from the earlier designs, we created OnSet and its constituent visual representations and interactive operations. One key property of OnSet is its ability to visualize large sets, that is, sets with cardinality in the hundreds of elements.

4.1 Visual Representation

To help explain the visual representation used in OnSet, we provide an example. Consider 5 sets that contain 10 elements each. For simplicity, let us assume that each element is present in only one set. This generates a master set (total list) of 50 distinct elements. In the OnSet technique, each set is represented not just by the elements that it contains, but by all 50 elements. The two categories (present or absent) of elements are, however, visualized differently to provide a distinction.

We depict each set as an m x n grid of locations. Each location can hold a potential element of the set. This grid is created from the master list of elements. Each element in this list is assigned a specific, unique location in the grid that is the same across all sets. In the visualization, each location is depicted as a square-ish rectangle. We call each of these locations a "pixel", leveraging an analogy to the screen pixels that make up a raster or bitmap on a display. Note, however, that one of these pixels in OnSet is likely implemented by a rectangular region of multiple, actual screen pixels. From this point forward, when we use the term "pixel", we are referring to the logical grid location within OnSet. Each pixel can take on one of two states, colored or blank. The pixel is colored if the element it corresponds to is present in the set, and blank otherwise. We chose blue as the default color for each pixel. We call a complete grid of pixels representing a set a PixelLayer. Figure 4 presents a PixelLayer view.

The dimensions of PixelLayers and the order of pixels within each are consistent across all sets. We choose the dimensions m x n of the matrix based on the following rules:

1. The product m * n is greater than or equal to the total number of distinct elements in the collection.
2. The ratio m / n is close to 1.

Using the two rules promotes a squarified grid, which improves the readability of the view. It is, however, likely that the total number of cells exceeds the total number of elements, in which case some cells in the grid are empty and not associated with any element. Each PixelLayer is annotated with two labels: one corresponding to the name of the set it represents and the other to the number of elements present in the set.

4.1.1 Pixel Ordering

The default order of the pixels in a set's PixelLayer is based on the sequence in which the elements appear in the original data or the master set. Using properties of the datasets can help to generate better configurations than would be created by simply appending elements to the sequence as they are found in the sets. For instance, the elements in a PixelLayer can be placed in such a way that the most frequently occurring elements are located on the top left and least occurring elements appear in bottom right. This frequency can be calculated based on either all the sets in the data or only those that are currently on the canvas. The latter has a downside, however, in that as each new set is introduced on the canvas, the configuration of elements in all sets is likely to change.

Another heuristic that can be used for positioning the elements is to employ an underlying property of the elements. For instance, this property can be a user-assigned value for each element based on some attribute or characteristic of the element. These values can be used as input for positioning the elements, such that the highest valued elements are at top left. If multiple such characteristics exist, the user can choose the appropriate characteristic from a menu to update the configurations on the fly.

4.1.2 Element Hierarchies

The underlying data can have additional properties that also can be used in the OnSet representation. For instance, if all the elements in the collection come from a hierarchy, this information can be used to group and place the elements within a PixelLayer. More specifically, all the elements within the same branch of the hierarchy can be grouped together within a sub-rectangular region within a PixelLayer. To help with placement of these groups, we utilize the

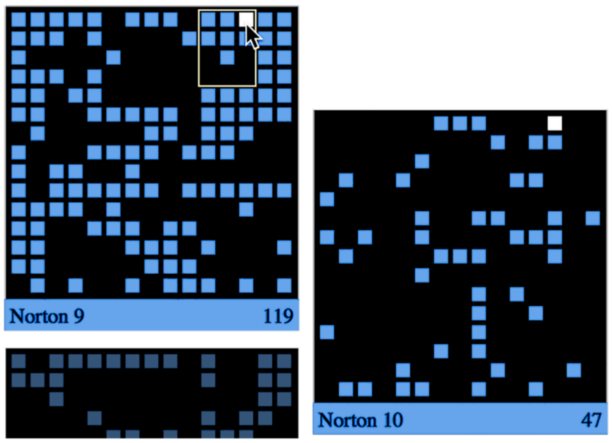

Fig. 6. Hovering over a pixel with the mouse highlights the same element in other PixelLayers (sets).

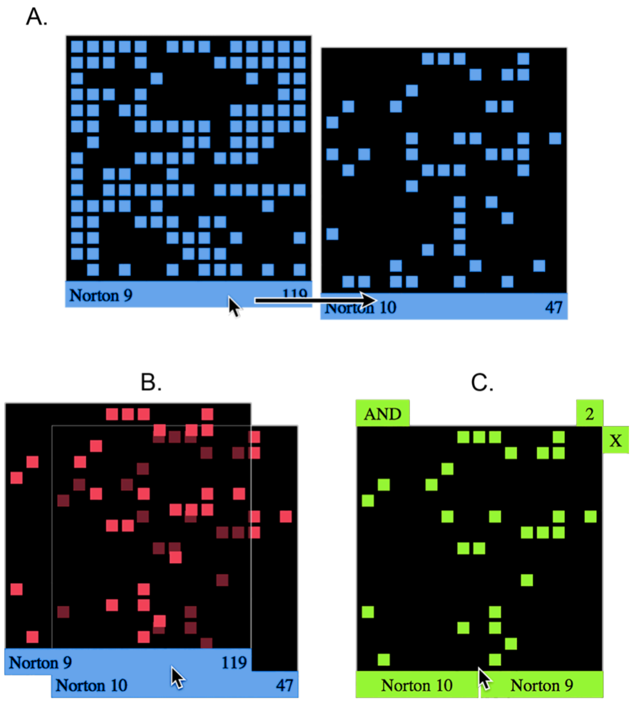

Fig. 7. Dragging and dropping a PixelLayer to create a new AND MultiLayer.

quantum treemap [5] technique. Similar to that done in the PhotoMesa system [4], we quantize the hierarchical groups within the PixelLayers using the number of elements in each hierarchy as input. We describe an example of this technique in the Use Case section that follows.

Once the groups of elements are laid out, the hierarchical structure is shown while hovering or brushing on the PixelLayers. Each hierarchical group is denoted by thin, off-white rectangles around the pixels in that group, as shown in Figure 5. Highlighting a pixel by hovering also highlights the bounding rectangle that depicts all the elements in its group. This behavior helps to gain a sense of how many elements from the hierarchical group also are present in the set. Brushing on the hierarchy also occurs when the user hovers over a pixel that is empty.

4.2 Interactive Operations

The OnSet visualization begins with a blank canvas for the exploration of the sets. In this subsection, we describe the interactive operations that the system provides.

4.2.1 Adding a PixelLayer

All sets that are present in the data are available in a list menu as shown in Figure 1. The user can import any set from the list by clicking the '+' button for that item. Doing this creates a new PixelLayer representing the set that is added to the canvas. Each PixelLayer that is subsequently added appears next to the previously added one. The PixelLayers appear in a sequential order, wrapping to the next row on the canvas when required. We chose to use a '+' button instead of a checkbox in each list item because each set can be added multiple times to the canvas. We discuss this feature in detail later.

4.2.2 Identifying Common Elements

Identifying elements that are common across different sets is one of the core tasks of set exploration, as discussed earlier. With OnSet, once the PixelLayers are laid out next to each other, it is relatively easy to identify common elements in two sets by visually inspecting the two layers. Since the position of an element is the same in all the PixelLayers, the elements that are common appear as blue pixels at the same position in each. However, for sets that are large, locating positions in different sets becomes more difficult.

To help with this problem, we utilize mouse hover to support this identification. Hovering over a blue pixel (element) in a PixelLayer makes it the active pixel, which is depicted by changing its color to white. Additionally, OnSet highlights the pixel in any other layer (set) that also contains the corresponding element. The user can then inspect the other layers for white pixels. To further simplify locating these layers, OnSet fades out all PixelLayers that do not contain the active pixel. Figure 6 shows the hover operation. The information panel at the top displays information about the element corresponding to the active pixel. Hovering over an empty element shows this information as well, but does not affect the items on the canvas in any way. This allows users to see information about the missing elements.

The mouse hover interaction provides a quick method for accessing all elements of a set individually and identifying ones that commonly occur across different sets. However, for tasks that involve comparing all the elements of two sets, a series of hover operations is likely to be time-consuming. For more direct comparison of two complete sets, we use a drag-and-drop action.

Each PixelLayer can be manipulated and positioned anywhere on the canvas by simply dragging it and dropping it elsewhere. However, if a PixelLayer is dragged and dropped directly on top of another layer, the two layers combine to form a composite PixelLayer, called a MultiLayer. (In the sections below, we abbreviate PixelLayer as PL and MultiLayer as ML for brevity.) A ML follows the same structure as an individual PL. However, instead of displaying the pixels (elements) of an individual set, a ML displays those pixels that are common to the two sets. In other words, a ML performs the equivalent of a logical AND operation on the corresponding pixels of the two PLs, where the value 1 indicates element presence and 0 indicates absence. OnSet draws pixels that are present in both PLs in green, with all other pixels appearing black.

Figure 7 presents the steps that are involved in this interaction. As Figure 7b highlights, the user receives preview feedback of the common pixels in the two layers when she drags one layer over another. She can then choose to confirm the combination by releasing the mouse-press or cancel it by dragging the PL back outside.

A MultiLayer is modeled in the same way as a PixelLayer. Hence, the interactions with a ML are similar to those on a PL. Mouse hover on a pixel in a ML highlights the same pixel in other PLs. Conversely, highlighting a pixel in a PL highlights the pixel in the ML if the element is contained in both the sets of a ML.

By default, a MultiLayer uses an AND operation to compare the corresponding pixels in the two PLs. However, MLs also support an

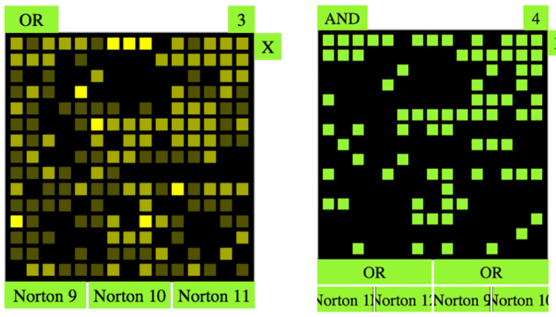

Fig. 8a. A MultiLayer OR with three sets. 8b. A MultiLayer AND of nested OR layers.

OR comparison operation. Switching the state to OR changes the color of pixels from green to yellow. Subsequently, each pixel assumes one of three states based on whether the corresponding element is present in neither of two sets, is present in one of the sets, or is present in both sets. These three states are depicted using a color gradient that goes from black to yellow. For example, if an element *x* is found in one of two sets, the pixel is given 50% yellow. If more PLs are added to an ML, an increasing number of different yellow shades are employed to denote the number of sets containing the element.

We use the term OR here but this operation does not operate purely like a logical OR. Instead, the OR operation here displays the percentages of sets containing an element. We use the terms AND and OR, rather than “intersection” and “union”, to provide a more natural mapping of the operation to a user's mental model and the questions they may be trying to address, such as “Are there elements that are found in both set A and set B?” and “Which elements are found in set A or set B?”

4.2.3 Comparing Sets

The MultiLayer representation supports easy comparisons of sets and is scalable to higher number of sets. Two types of scalability result:
A ML can be overlapped with other PLs to generate a higher-order ML. These higher-order MLs carry over the state from the original ML. That is, adding a PL to a ML in the AND state creates a new ML in the AND state. In this new ML, the pixels are formed by performing an AND operation on the corresponding pixels of all PLs that form the ML. Similarly for OR state, the pixels are formed by performing an OR operation and the colors assigned correspond to the number of PLs in which the corresponding element appears. Figure 8a illustrates this feature.

A ML can also be overlapped with other MLs to generate a higher-order ML. However, this operation differs from the previous case in a few ways. First, the higher-order ML always assumes an AND state irrespective of the states of the individual MLs, although the state can be changed to OR later. Second, the AND or OR operation only considers the pixels of the original MLs that are combined and not the underlying PLs that made up the MLs. Figure 8b illustrates this feature.

Expression Tree
For each of the two types of MLs, the value of a pixel is determined by evaluating an expression tree. A complex expression is generated between the constituting PLs depending upon the type of ML generated.

1. Dragging a PL onto another PL: Creates a new expression containing the two sets.
2. Dragging a PL onto a ML: appends the PL set to the expression at the top level of the expression tree of the ML.
3. Dragging a ML onto another layer: Creates a new expression containing the two expressions, which in turn are nested as a new level in the expression tree.

To explain the third type, we will walk through creating the expression (A || B || C) && (D || E) for sets A, B, C, D, and E. (The symbol || denotes OR and && denotes AND.) The first step is to create the nested expressions. Dropping set B on to set A forms the A && B composite ML. Dropping set C onto this ML then forms A && B && C. Clicking the operation label changes the expression to A || B || C. The same process is used to create the D || E composite. Finally, the D || E composite ML is dragged onto the A || B || C composite ML, forming the desired operation. A pixel is shown if the element is found in sets A, B, or C, and found in sets D or E.

Labels and Buttons
Visual elements are used on the MLs to display information about their various properties. The labels on the bottom reflect the structure of the expression being shown. These labels appear in one or more levels depending on the current configuration of the ML. Labels for nested MLs are appended to the labels for the parent expression, forming an icicle plot of the hierarchy. As space quickly becomes limited and labels are obscured, hovering over a label displays a tooltip of the full set name. Dragging a label from within a ML removes the corresponding PL node from the composite, creating a new layer with the removed item. If that node is a single set, a single PL will be created. If that node is another expression, a ML will be created. Only the top-level components in the ML can be dragged to encourage decomposing in an orderly manner.

The top left of the ML contains a label describing the current state of the expression, i.e. AND or OR. This label can be clicked to toggle the states. Doing so evaluates the new expression tree and updates the color of the pixels in the layer. A label on the right shows the count of the sets that currently constitute the ML. Finally, in addition to dragging labels to separate the individual PLs, an ML can be quickly disintegrated by clicking on the "X" button that is available on the top right corner. This splits the top-level expression into all of its components, forming new PixelLayers or MultiLayers, as required, and placing them on the canvas.

4.2.4 Similarity

Another capability that is relevant for set comparisons is a similarity calculation. Set similarity can be defined in multiple ways. In a basic form, the similarity of two sets can be calculated from the number of elements that are present in their intersection. This metric favors sets with more elements, however. When the master set of elements is known, the similarity of two sets can be defined by using the number of elements that match via either common presence or absence (i.e. they satisfy the logical equality or XNOR operator). In other words, the similarity is derived from the number of elements that are commonly present in both sets plus the number of elements similarly absent in both sets, given the context of the master element list.

OnSet, by default, takes the latter approach to calculating a similarity metric. If an element is present in two sets, their similarity score is increased by one. Likewise, if an element is absent from both those sets, the similarity score is increased by one. The score is then normalized by dividing it by the number of elements in the master set. By taking into consideration the elements missing from both sets, this similarity score closely reflects the visual encoding of the PixelLayers, where both present and absent set elements are shown to the user. However, in addition to the default similarity metric, the Jaccard index may be used. The Jaccard index is defined as the cardinality of the intersection of two sets divided by the cardinality of the union of those two sets. It does not give as much weight to missing elements as the default metric does, but it is more sensitive to subtle differences in small sets

The similarity score is calculated for every pair of PixelLayers on the canvas, including MultiLayer pairs and PixelLayer/MultiLayer pairs. Because every layer, regardless of type, represents a set, the calculation is performed in the same way between all pairs. The similarity is then represented by a band that connects the two layers. The thickness of the band is directly proportional to the similarity metric. The normalized score is easily scaled to the maximum and minimum thickness. OnSet's default start state does not show the bands, but the user can display them by selecting the "Similarity" checkbox on the right of the user interface. In addition, a dropdown menu allows the user to select the similarity metric to employ.

When many PLs are shown on the canvas, the similarity bands can help users find interesting set relationships to focus on and explore. The bands also are interactive. Hovering over a band displays details about the similarity between the connected PLs in the information bar at the top of the interface. In addition, all the elements that are present in both PLs connected by the band are highlighted, while the PLs that are not connected to the band fade out. This feature is shown in Figure 9. Finally, clicking on the band "selects" the pair, keeping their common elements highlighted while the user interacts with other parts of the interface. Clicking elsewhere on the canvas clears the selection.

4.2.5 Brushing and Locating Sets and Elements

The list of sets in the right of the interface (see Figure 1) provides information about the sets loaded into the system. Each item in the list contains the set label as well as a horizontal bar indicating the cardinality of the set. The cardinality is constrained by the elements found in the master list of elements used to render the PixelLayer grids. It is possible that the true "universal" set of all elements found in the data is larger than the master list. However, elements that are not found in the master list are ignored. The set list supports sorting sets by labels alphabetically and by cardinality. A search box can be used to search for sets with a certain label, as well. The number of sets found for the current search (or total number of sets if no search filter is in place) is displayed at the bottom of the list. Brushing an item with the mouse in the set list fades out any PLs on the canvas that do not correspond to that set. This allows users to quickly locate any instances of the set on the canvas.

In addition to the set list, OnSet provides a list of all the set elements. This list also supports searching and brushing. If an element is brushed in the list, all of that element's present pixels in PLs on the canvas are highlighted, turning white. Any layers that do not contain a pixel corresponding to the element fade. This allows users to search for specific elements and see what sets contain them.

4.2.6 Zoom and Pan

The canvas supports zooming and panning to allow better arrangements of the visible layers. This is especially important as the number of layers increases. Clicking and dragging on an empty space in the canvas will move the contents in the direction of the drag. The mouse wheel may be used to increase or decrease the magnification scale. An indicator in the top right of the interface shows the current zoom level, with buttons to zoom in and out, as well.

4.3 Implementation

The prototype OnSet system runs in a web browser and is written entirely in JavaScript, using the D3 visualization toolkit [6]. The right-side interface controls were coded using the jQueryUI widget library. The system can be accessed at the following link: http://www.cc.gatech.edu/gvu/ii/setvis.

4.4 Task Support and Scalability

Earlier we noted the different types of tasks on set-typed data identified by Alsallakh et al [3]. With respect to these tasks, OnSet supports element-related tasks such as finding the elements belonging to a specific set or finding the sets containing a specific element, but not operations such as finding elements in sets A and B

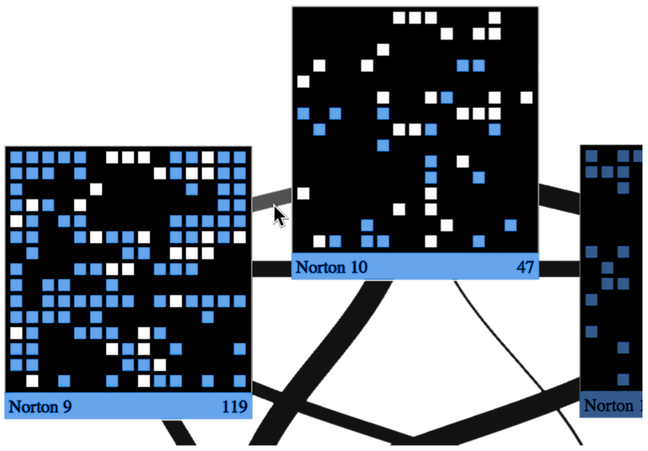

Fig. 9. OnSet shows the similarity of two sets via the thickness of a band between them. Hovering over a similarity band highlights the common elements between two sets.

but not C (without the NOT operator), finding elements exclusive to a set, or filtering out elements. For tasks related to sets and set relations, OnSet supports different combinations of intersection and union operations, as well as estimations of these cardinalities, along with comparison of set similarities. Of course, interaction is required to perform these operations. It is difficult to perform set relations tasks such as finding the specific set intersections resulting in some other set of elements. As discussed earlier, OnSet does not support attribute-related tasks on elements, but in Section 6 we discuss how such tasks could be addressed within the OnSet framework.

Scalability can present challenges for a system such as OnSet in multiple ways. First, as the number of elements increases, the representation of a set may need to be so large that not many sets can be shown and/or the size of each element becomes too small to adequately differentiate and employ. Alternately, the interactive behavior of the application may degrade as more elements and/or sets are added. Via trial use, we believe that the current technique can well represent PixelLayers with a cardinality of roughly 2000 (45x45), still giving each element adequate size (pixels) and without losing significant interaction ability. Further, for even larger sets and numbers of elements, a user can employ the system's zooming and panning capabilities to condense the size of PixelLayers in order to accommodate more per canvas.

For situations when the individual sets are large but sparsely populated, however, the technique may be less effective because so much space is devoted to the sets and not the data elements. Again, through trial use, we have found the technique to work better for sets having more than about 20% of the elements present.

5 CASE STUDIES

To help the reader understand the utility of the OnSet technique, we describe two case studies employing OnSet to understand set data. The case studies help in identifying the interactions that support effective exploration of data with the visualization technique. In the first case study, we examine the domain of biochemical data. In the second, we explore calendar and event data.

5.1 Whale Sharks and Blood Samples

Whale sharks are the largest species of fish, growing to a length of up to 13 meters. They are on the list of vulnerable species and, as a result, marine biologists closely monitor their population and wellbeing. A recent study [8] used NMR and mass spectrometric methods ("metabolomics") to analyze whale shark blood as part of this health monitoring effort. It generated large tables of chemical data that were difficult to analyze and understand. We have been able to procure data for whale sharks that were or are currently resident at the Georgia Aquarium in Atlanta, GA.

Biologists primarily use this data to gauge the health of the sharks by analyzing and comparing the samples across days or weeks, or by comparing healthy and unhealthy individuals. Profiling the samples helps in identifying any toxicity assessment as well as tracking any environmental effects caused as a result of individual's interactions with their surroundings. Their intention is to find anomalies or trends in the chemical composition and relate that to the diet and health status of the sharks. Doing this based on chemical names, rather than quantitative measures such as concentration, presents a unique analysis challenge. In many cases, biologists approach this data without specific questions but seeking to explore and discover insights. Additionally, they want to detect trends in the blood samples and identify characteristics that may predict health issues.

We can model these requirements into specific questions of the following type:

1. Does a blood sample contain the toxic compound Ciguatoxin?
2. Has the compound been present in the shark's blood during previous periods as well?
3. Is the compound present in any other shark's blood? If so, what other common compounds also are present?
4. Are there patterns of chemicals that are consistent in shark X but only recently appear in shark Y?

Each blood sample is described in terms of all the biochemical compounds that are detected by spectrometry. These compounds only appear as names without their corresponding concentrations. Our data included samples for ten whale sharks, with an average of about 5 samples per shark, each taken on different days. Every sample contains between 150 and 300 compounds, with a total of about 1100 distinct compounds across all the samples. We can model each blood sample as a multi-variable set of binary-state data. A sample can be modeled to have 1100 distinct possible compounds, with each sample being defined by both the compounds that are present in it and those that are absent. In addition to detecting the presence of particular compounds in the samples, it is important to clearly observe the absence of specific compounds as well.

With the help of an Aquarium biologist, we analyzed the 1100 distinct compounds and identified a set of the 225 most frequently occurring compound and used them as the master set. We thus used a 15 x 15 grid for representing a sample as a PixelLayer. The compounds also belong to certain chemical classes, such as Aldehydes or Amino Acids. This information was used to group the pixels in the PixelLayer in a quantum treemap [5] approach, though the layout was manually tweaked to avoid empty spaces that can be produced by the technique. Blood samples were imported into the application (Figure 1) with a label containing the shark's name and a sequential number. A higher number indicated a more recent sample. The data associated with the sample was a large list of compound names. All of the operations described earlier for OnSet are available in this application we called AquaViz. An early version of the system was described in [18].

Consider the following scenario. One of the whale sharks, Ralph, has been ill. Both Ralph and another shark, Norton, have recently been put on a new diet. Biologists want to identify any chemical compound(s) related to Ralph's illness and determine if the illness may spread to Norton through their shared diet and eliminate any items that are known to introduce the compound(s). To do so, they run AquaViz.

Using the samples list, they are able to locate the three most recent samples for both Norton and Ralph and are surprised to see that Ralph's blood contains fewer compounds in it than Norton's, as indicated by the horizontal bars below the sample names. This interesting discovery helps motivate them to dig deeper into the data. They add the six samples to the canvas. Looking at the PixelLayers, it is clear that Norton's samples have a higher number of compounds than Ralph's. The biologists drag each shark's samples together to form two MultiLayer representations, one containing Ralph's samples and one containing Norton's. In the AND state, the MLs seem to be very similar. Switching the operation to OR, however, reveals a large cluster of slightly faded compounds in Norton's layer that are missing from Ralph's. Hovering over the elements, the biologists are able to see that Norton's blood contains important amino acids that Ralph's does not.

The biologists add the most recent samples for the other sharks and turn on the bands showing the similarity metric for each layer pair. The bands connected to the Ralph ML are all similar in weight but smaller than the bands connected to the Norton ML. Most bands connected to the Norton ML are similar to each other, too. However, the band connecting Alice's layer to the Norton layer is quite large. Hovering over the band confirms that the two layers share a large number of compounds. Interested in this relationship, the biologists remove all other layers from the canvas, then break apart the Norton ML back into the individual sample PLs. The similarity bands update to show that only the two most recent samples share much in common with Alice's data. Using this information, the biologists review camera footage of the past two days and discover that Norton was accidentally fed Alice's food. They conclude, however, that this mistake was a good one. The new diet for Ralph and Norton has been ineffective and Norton may have fallen ill, as well. The biologists determine that both shark's diets should be switched to one similar to Alice's.

As seen in this scenario, the interactive, direct manipulation of the samples and compounds provides the user with a "hands on", flexible environment for investigating different combinations of samples and making sense of the data found in them. The user can explore similarities and differences across samples, as well as patterns and peculiarities across multiple samples simply by dragging them around the canvas.

5.2 Calendar Visualization

We also utilized the OnSet technique for the purpose of task and event scheduling. Consider the following scenario: Jonathan, who is in HR at a web-development company, wants to discuss the recent changes to the hiring process introduced by the company management with the teams he overlooks. Accordingly, he decides to schedule a meeting with all the team leads as well as at least one program manager from each ongoing project. He sends a meeting request to all the people for a particular day and time. He immediately receives feedback from a team lead stating that the time does not work for her and that he should create a Doodle poll.

The above scenario describes a situation that we often face in our daily lives. We frequently use polls in such situations to find an optimum time slot. However, these polls have a few limitations. First, the host needs to specify an exhaustive list of options at the start. Additionally, each participant is also required to individually answer the poll. Finally, the application does not provide a method to create meta-configurations. For instance, in the above scenario, the host cannot configure the poll so that the presence of all team leads is mandatory, but only one of the program managers needs to be present.

We observed that by modeling the calendar data of each participant in a specific manner, we could utilize our set visualization technique for the above-mentioned scheduling task. The data would be modeled so that each person corresponds to a set, with the elements within the set being the time slots, both available and unavailable. Comparing the corresponding sets could highlight time slots that are available across participants.

Various schemas can be used to present the calendar data, since the structure of a PixelLayer is modular. A set could be structured to present the data for a week, a month, or a year. For each, further possible configurations exist. For instance, data for a year can be represented as 'number of weeks in a year' x 'number of days in a week', i.e. a 52 x 7 matrix. Alternately, a year also can be represented as 'number of months in a year' x '(max) number of days in a month', i.e. a 12 x 31 matrix. Which scheme is appropriate depends on the intended task. In our scenario, the task relates to scheduling a meeting in a particular week. As a result, we use one week as the unit of representation. We model a business work-week

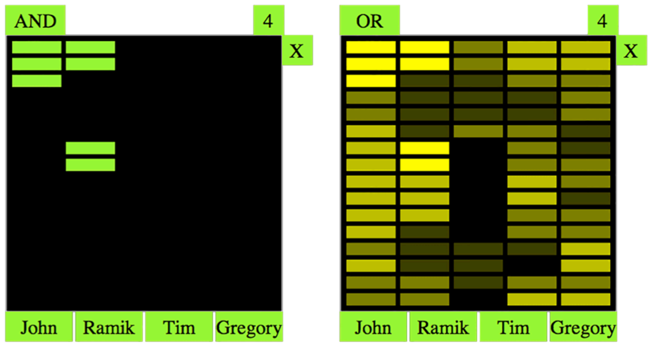

Fig. 10. PixelLayers from the CalViz application.

as 5 days of work with 8 hours of work each day. Accordingly, the layer consists of 5 columns and 16 rows, with each column representing a day and each row representing a half-hour slot.

We developed an application called CalViz that implements the OnSet visualization paradigm. Figure 10 shows a view of PixelLayers in the system. Here, the data encoded as elements is the inverse of the events retrieved from a calendar. Events are those slots in which a person is busy. However, we show the inverse, or all slots in which the person is available, in the PixelLayer.

The CalViz interface is similar to the AquaViz interface with the list of sets now revealing the names of the people whose calendar information is accessible. In our scenario described earlier, Jonathan imports a person's calendar information for a week onto the canvas and the PixelLayer shows up as a 16 x 5 matrix. However, the present (blue) pixels represent those time periods in which the person is available. This is different from the schema of highlighting elements that are present in the set, as we used in all the earlier examples in the article. The main reason for doing so is that the task driving the interface, namely identifying slots of time when multiple people are available, is paramount. Once multiple people are added, the available free time is found by locating the colored pixels.

The CalViz visualization shows data for one week at a time. The selected dates are highlighted at the top of the interface. A different week can be selected by using the left and right buttons or by opening a calendar input menu that permits selecting a week at a time. Changing the selected week affects all the samples that are thereafter loaded on the canvas. Changing the selected week, however, does not update the data for the samples that already have been added. The purpose behind this design decision was to support a particular type of task. Consider the situation where a user wants to schedule a weekly meeting with a person for the 4 upcoming weeks. To do so, the user would need to find a slot that works for the person in all the 4 weeks. CalViz allows a user to import the same person multiple times. Changing the selected week before each import, and then comparing the four layers by overlaying them on each other, the user would be able to see only those time slots that are available in all the four weeks.

6 Conclusion

In this article, we introduce OnSet, a technique for visualizing large-scale binary set data. OnSet is a pixel-matrix based representation technique that models each set by the elements that it both contains and does not contain. The technique employs direct manipulation interaction and visual highlighting to support identification of commonalities and differences as well as membership patterns across different sets of elements. The technique is scalable and easily supports sets with cardinality in the hundreds. We discuss the applicability of the technique across different domains with the help of two use cases. In the first use case, we utilize spectrometry data from blood samples of whale sharks and identify the common compounds that occur across samples. In the second use case, we discuss a task-scheduling scenario where participants' calendar data is used to find available time slots.

The OnSet technique differs from existing set visualization techniques in a number of key ways. First, the technique does not use only one visual representation of each data element. Instead, a particular element occupies a specific position within each set's representation. This obviates the need for irregular shapes and contours for set boundaries. It also provides visual evidence showing the absence of an element in a set as well as presence. The OnSet technique also does not encode subset relationships, similarities, and differences solely through visual imagery. OnSet instead utilizes direct manipulation stacking of set representations to perform different set operations. Complex composite operations can be iteratively constructed and de-constructed.

In terms of future work to be addressed, one main shortcoming of the OnSet technique is its limitation to binary set data, that is, its inability to also show attributes of the data set elements. For instance, consider a data set of people. Each has a hair color, eye color, height, weight, and so on. OnSet is able to show large sets of these people and communicate individual's presence or absence in these sets, but it currently cannot portray information about the people's attributes. To address this issue, each categorical value, e.g., blonde hair, could become its own set in OnSet. People with that attribute value then would have their pixel positions colored in the "blonde hair" PixelLayer. Attributes could even be combined via OnSet's composition so that, for instance, a set showing blonde-haired, green-eyed people could be created. Of course, quantitative attributes would still be problematic. Still, this approach would seem to be more beneficial than other techniques that use color to indicate different element attributes and quickly run out of colors.

Additional future work involves adding new capabilities to the system. Creating logical NOT and "minus" operations would be helpful as would allowing other sorting orders of elements, such as by frequency. Showing "empty" (where no element resides) cells at the end of a PixelLayer differently, improving the labeling, and showing counts of MultiLayer elements all would be beneficial.

Ultimately, creating coherent, comprehensible visualizations of large sets that share multiple elements is simply a very complex task! Researchers have developed a wide variety of visual representations to do so, and many become complex very quickly, making visual interpretation a challenge. In OnSet, we address this problem by using interaction to show many set operations that other techniques try to illustrate statically. OnSet's use of interaction in this way provides both advantages and disadvantages. Focusing first on the disadvantages, the technique requires the user to interact, thus the burden of performing meaningful, enlightening operations is on the user. Key set relationships are not laid out in front of the user for browsing. Conversely, we feel that the key advantage of OnSet's visual representation and use of interaction is that it provides a visualization that is more simple and comprehensible than most other techniques while still providing a great deal of power and utility. The rectangular PixelLayer model with specific element positions and the direct manipulation stacking of layers are concepts that can be easily learned and interpreted on the screen. We have demonstrated the technique to many other researchers and they grasp its basic methodology and operations very quickly. OnSet supports a diverse set of analytic tasks on large sets of elements and it does so in a manner that can be quickly understood and interpreted.

Acknowledgments

The authors thank Tim Dwyer for his in-depth discussions, feedback, and advice about the technique. This work was supported by the Georgia Aquarium, the DARPA XDATA program, and the National Science Foundation via awards CCF-0808863 (FODAVA) and IIS-1320537.

REFERENCES

[1] B. Alper, N. Henry Riche, G. Ramos, and M. Czerwinski. Design Study of LineSets, a Novel Set Visualization Technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12): 2259-2267, 2011.

[2] B. Alsallakh, W. Aigner, S. Miksch, and H. Hauser. Radial sets: Interactive visual analysis of large overlapping sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12): 2496-2505, 2013.

[3] B. Alsallakh, L. Micallef , W. Aigner, H. Hauser, S. Miksch, and P. Rodgers. Visualizing Sets and Set-Typed Data: State-of-the-Art and Future Challenges. In *State-of-the-Art Proceedings of the EuroGraphics Conference on Visualization (EuroVis '14)*, to appear.

[4] B. Bederson. PhotoMesa: A Zoomable Image Browser Using Quantum Treemaps and Bubblemaps. In *Proceedings of ACM UIST '01*, pages 71-80, Nov. 2001.

[5] B.B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. *ACM Transactions on Graphics*, 21(4): 833-854, 2002.

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on. Visualization and Computer Graphics, 17(12): 2301-2309,* 2011.

[7] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6): 1009-1016, 2009.

[8] A.D.M. Dove, J. Leisen, M. Zhou, J.J. Byrne, K. Lim-Hing, et al. Biomarkers of Whale Shark Health: A Metabolomic Approach. *PLoS ONE*, 7(11): e49379, 2012. doi:10.1371/journal.pone.0049379.

[9] J. Flower and J. Howse. Generating Euler Diagrams. In *Proceedings of the 2nd International Conference on Diagrammatic Representation and Inference (DIAGRAMS '02)*, pages 61-75, 2002.

[10] W. Freiler, K. Matkovic, and H. Hauser. Interactive Visual Analysis of Set-Typed Data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6): 1340-1347, 2008.

[11] G. Furnas. Generalized Fisheye Views, In *Proceedings of ACM CHI '86*, pages 16-23, 1986.

[12] J. Howse, F. Molina, J. Taylor, S. Kent, and J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. *Journal of Visual Languages and Computing*, 12(3): 299-324, 2001.

[13] D.A. Keim, P. Bak, and M. Schäfer. Dense Pixel Displays. *Encyclopedia of Database Systems*, pages 789-795, 2009.

[14] B.H. Kim, B. Lee, and J. Seo. Visualizing set concordance with permutation matrices and fan diagrams. *Interacting with Computers*, 19(5-6): 630-643, 2007.

[15] R. Rao and S.K. Card. The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. In *Proceedings of ACM CHI '94*, pages 318-322, 1992.

[16] P. Rodgers. A survey of Euler diagrams. *Journal of Visual Languages and Computing*, 2013, to appear.

[17] N. H. Riche, T. Dwyer. Untangling Euler Diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6): 1090-1099, 2010.

[18] R. Sadana, A. Dove, J. Stasko. Whale Sharks, Boolean Set Operations, and Direct Manipulation (Poster). IEEE Information Visualization 2013 Conference, 2013.

[19] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of IEEE VL '96*, pages 336-343, 1996.

[20] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualization of overlapping sets. *Computer Graphics Forum*, 28(3): 967-974, 2009.

[21] J. Yang, D. Hubball, M.O. Ward, E.A. Rundensteiner, and W. Ribarsky. Value and Relation Display: Interactive Visual Exploration of Large Data Sets with Hundreds of Dimensions, *IEEE Transactions on Visualization and Computer Graphics*, 13(3): 494-507, 2007.

[22] J. Yang, Y. Liu, X. Zhang, X. Yuan, Y. Zhao, S. Barlowe, and S. Liu. PIWI: Visually Exploring Graphs Based on Their Community Structure, *IEEE Transactions on Visualization and Computer Graphics*, 19(6): 1034-1047, 2013.

[23] J.S. Yi, R. Melton, J. Stasko, and J. Jacko. Dust & Magnet: Multivariate Information Visualization using a Magnet Metaphor. *Information Visualization*, 4(4): 239-256, 2005.