Smooth Graphs for Visual Exploration of Higher-Order State Transitions

Jorik Blaas, Charl P. Botha, *Member, IEEE*, Edward Grundy, Mark W. Jones, Robert S. Laramee, *Member, IEEE*, and Frits H. Post

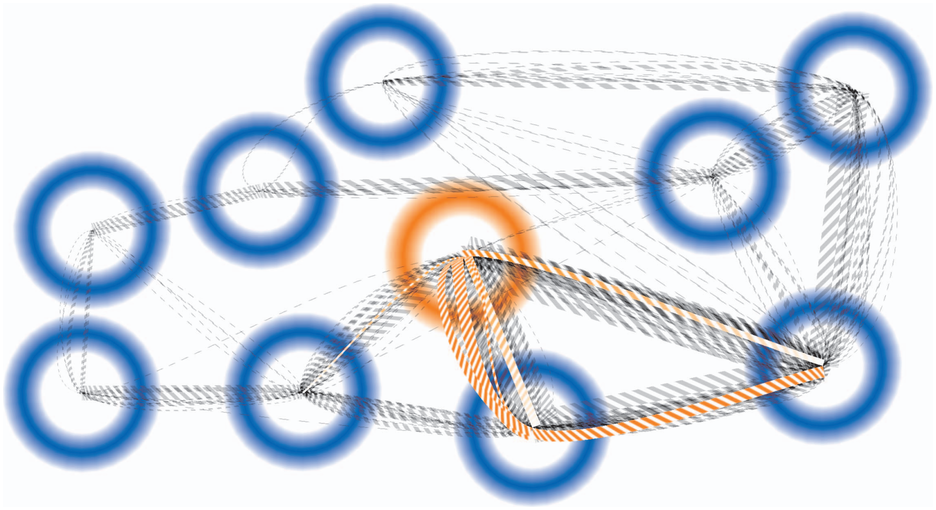

Fig. 1. A smooth graph representation of a labeled biological time-series. Each ring represents a state, and the edges between states visualize the state transitions. This graph uses smooth curves to explicitly visualize third order transitions, so that each curved edge represents a unique sequence of four successive states. The orange node is part of a selection set, and all transitions matching the current selection are highlighted in orange.

Abstract—In this paper, we present a new visual way of exploring state sequences in large observational time-series. A key advantage of our method is that it can directly visualize higher-order state transitions. A standard first order state transition is a sequence of two states that are linked by a transition. A higher-order state transition is a sequence of three or more states where the sequence of participating states are linked together by consecutive first order state transitions.
Our method extends the current state-graph exploration methods by employing a two dimensional graph, in which higher-order state transitions are visualized as curved lines. All transitions are bundled into thick splines, so that the thickness of an edge represents the frequency of instances.
The bundling between two states takes into account the state transitions before and after the transition. This is done in such a way that it forms a continuous representation in which any subsequence of the timeseries is represented by a continuous smooth line. The edge bundles in these graphs can be explored interactively through our incremental selection algorithm.
We demonstrate our method with an application in exploring labeled time-series data from a biological survey, where a clustering has assigned a single label to the data at each time-point. In these sequences, a large number of cyclic patterns occur, which in turn are linked to specific activities. We demonstrate how our method helps to find these cycles, and how the interactive selection process helps to find and investigate activities.

Index Terms—State transitions, Graph drawing, Time series, Biological data.

1 Introduction

One of the common ways to visualize state transition sequences is by using graphs. Each node represents a state, and an oriented edge between two nodes represents a transition between those two states. For the exploration of time-series label data, such a graph can be constructed by examining all succeeding pairs of states and generating a set of edges between the nodes representing them.

While these graphs give a good overview of the transitions between states, one important aspect is lost in the visualization: the context in which these transitions occur is not visible. Figure 2 shows how the first-order transition graph may ambiguously represent multiple underlying sequences. The leftmost graph could either correspond to the sequence ABCABCABC..., CDECDECDE..., i.e. multiple repetitions of each of the triangles as shown in the rightmost graph, or the sequence ABCDECABCDECA..., i.e. all states in one long sequence passing multiple times through C as shown in the middle graph. One of our goals is to visually disambiguate these two situations. In the following sections, we elaborate how this can be done by taking into account higher-order state transitions when drawing the edges. The middle and rightmost graph representations in figure 2 show how our method uses curved edges to emphasize the order in which the transitions occur, which gives each of the two sequences a unique visual representation.

• Jorik Blaas, Charl P. Botha and Frits H. Post are with the Data Visualization Group, Delft University of Technology, NL.
• Edward Grundy, Mark W. Jones and Robert S. Laramee are with the Visual Computing Group, Swansea University, UK.

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

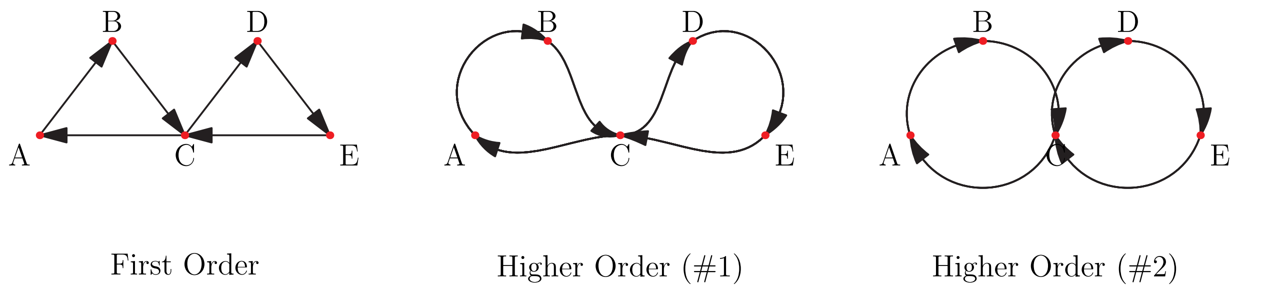

Fig. 2. A graph representation of a state transition sequence. The leftmost figure uses classical first order transitions, making it impossible to distinguish the sequence ABCABCABC..., CDECDECDE... from the sequence ABCDECABCDECA.... The rightmost figures take into account higher-order transitions to visualize the order in which the state transitions occur.

1.1 Problem Domain

Our method is designed specifically to handle labeled time-series data. In such data, a set of labels is defined, and each point in time is assigned one of these labels. Each of these labels can be seen as a state, and a change of state then forms a transition. One differentiating characteristic of these time-series data is that it contains information on the order in which state transitions occur. This also implies that statistical correlation can exist between the sequence of states taken to arrive at a node, and the transition taken to leave that node.

We want to be able to visualize these oft-occurring sequences of states, and we want to emphasize the surrounding conditions in time under which these occur.

For example, the state C in the sequence ABCDECA... (see Figure 2), is visited in two different contexts. It can be reached either from state B, or from state E. The subsequent state is determined completely by the previous state, as arriving from B means the next state will be D, and arriving from E the next state will be A.

This correlation between the incoming state and the outgoing state can be put into a broader perspective by looking at so called higher-order transitions.

1.2 Higher-Order Transitions

A standard first order state transition is a sequence of two states that are linked by a transition. A higher-order state transition is a sequence of three or more states where the sequence of participating states are linked together by consecutive first order state transitions. Our definition of a higher-order state transition is derived from that used with higher-order Markov models.

Representing the conditions, or context, under which transitions occur is an important task, as it leads to insight into the temporal linkage between the corresponding states. In other words, the fact that a given state change depends on the previous state change and has an effect on the following state change plays in important role in exploring such phenomena.

We have designed a visual method for representing these higher-order state transitions. In section 3, we present a representation that visually disambiguates the higher-order transitions, providing a *smooth graph* representation for visualizing state transition sequences.

A major point of novelty is that the presented method uniquely represents each unique higher-order transition with a curved line in such a way that any chain of subsequent transitions is visualized in a smooth and continuous fashion. This makes it not only easy to pinpoint frequently occurring transitions, but it also helps to identify the set of state transitions that lead to them by implicitly visualizing the temporal context.

2 Related Work

A state transition graph represents a system of states and state changes. Each state is represented by a vertex and each state change by a directed edge. State transition graphs are generally represented by simple node-link diagrams, where each node represents a state and each link a state transition [13].

These node-link diagrams have been extended with 3-D layout algorithms [11] and also with intelligent 3-D positioning of edges and nodes that better shows the hierarchy and organization of the state transitions [19, 18]. Taking a different approach to coping with highly complex node-link diagrams, Leuschel et al. presented techniques to reduce the complexity of the underlying state transition graphs by merging nodes [12].

Holten [8] proposes *Edge Bundles* for visualizing hierarchical data, where adjacency relations are bundled together using B-spline curves. He also shows that such a method can be combined with any of the existing major tree visualization techniques. Cui et al. [1] also cluster edges within graphs and encode additional information by using color and opacity. They also employ animation to show intermediate transitions from the original graph to the edge clustered graph, and to allow different levels of detail. They smooth polylines to ensure visually pleasing paths through the graph. Another example of using curves for graph visualization is given by Eppstein et al. [2] where they visually identify bicliques in a graph by depicting them using bezier curves such that each curve passes through the central point for the biclique. Although these three methods use smooth lines, they do not use any continuity constraints to represent higher-order sequences as we do here.

Continuity has long been identified as important to human perception, and Field et al. [3] demonstrate the ability of the human visual system to successfully follow continuous paths even against randomly oriented background paths, or with partially obscured paths. Curves have been used in parallel coordinates by Theisel [16] to show correlations in non-adjacent axes, and by Graham and Kennedy [5] to distinguish axis crossing visually through curve continuity.

Ham and Rogowitz [17] carried out a study where participants created their own graph layouts which could then be measured to try to evaluate the importance of various visual features. Through this process they are able to provide recommendations for graph drawing algorithms.

Also related to this work are visualization techniques for biological sensor data including Ware et al. [21] where accelerometer data was visualized using TrackPlots, which enabled scientists to check the theory that whales roll onto their sides for specific prey capture, and Grundy et al. [6] where spherical plots, spherical overlays, spherical histograms and a posture state graph are shown to be effective at leading to biological understanding

Pretorius et al. [14, 13] presented a unified approach to visualize highly complex state transition graphs, employing node-link diagrams to visualize a hierarchical clustering of the different states, the bar tree to visualize the number of occurrences of states and finally an arc diagram to visualize the actual state transitions. A method was also presented whereby multivariate graphs could be more explicitly explored: Nodes are arranged in a source group and a target group, and all edges are shown in between [15] and queries can be interactively performed. This method was also applied to state graph visualization. Herman et al. [7] provide a good survey of graph drawing techniques specifically aimed at information visualization.

All of the mentioned techniques have as one of their major goals

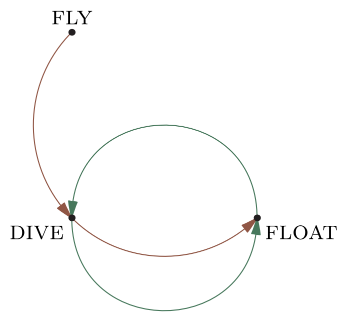

Fig. 3. Smooth graph representation of a set of two state sequences: FLY, DIVE, FLOAT in red and FLOAT,DIVE,FLOAT in green. Note the duplicate edges between DIVE and FLOAT

successfully coping with the high complexity of most state transition graphs. However, none of them take into account higher order state changes, i.e. state changes where the context, the states before and after the current state change, plays an important role. None of the techniques are able to visualize or represent these higher order state changes in any way.

The method that we present in this paper explicitly represents higher order state changes, thus enabling the exploration of state transition graphs where the context of a state change is important.

3 Method

A number of Gestalt principles exist that are applicable to the design of visual representations, as discussed by Ware [20]. In the design of our visual representation we focus on three aspects in particular: continuity, connectedness and closure. The principle of continuity states that elements that make up a smooth and continuous shape are more likely to be perceived as a visual entity. The connectedness principle entails that a relationship can be expressed by graphical objects by for example connecting them with lines. This principle is one of the prime principles of node-link diagrams, and a common method for visually representing relationships between entities. Closure implies that closed shapes are likely to be perceived as a single entity. We want to make sure that closed shapes likely occur where interesting patterns emerge, so as to focus on the cyclic patterns.

3.1 Design goals

Based on the principles above, we have formulated two main design goals. Firstly, each higher-order transition should be represented uniquely. Secondly, the visual representation should be continuous and smooth.

3.1.1 Uniqueness

When each higher-order transition is uniquely represented visually, each state in the transition influences the form, so that the combination of all states in the transition together uniquely identifies the shape. Since a first-order transition can occur as part of several higher-order transitions, this requires that edges are duplicated.

Figure 3 illustrates how this applies to a real-world example. Marine birds that dive have two different types of diving behavior, they either dive from flight, or they dive while swimming. In the analysis of our labeled time series, distinguishing between a dive that is initiated from flight and one that is initiated from a swimming position is a frequently occurring task. If we take into account second-order transitions, then the DIVE, FLOAT transition is part of the two second-order transitions FLY, DIVE, FLOAT and FLOAT, DIVE, FLOAT.

Instead of having to query the state sequences explicitly for the two patterns, our goal is to incorporate the higher-order information into the visualization, making a visual distinction between the two occurrences.

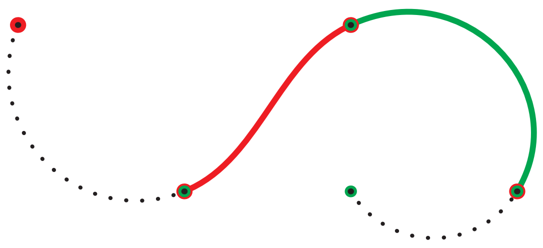

Fig. 4. Smooth spline interpolation of third order transitions, using a fourth order spline. Each of the colored segments is drawn by fitting a smooth curve through the four points of the same color.

3.1.2 Continuity

As the transitions that we visualize are all part of a longer sequence, we also want to make this clear in the visual representation. We constrain the visual representation of the edges in such a way that the longer sequence of transitions that links these edges together still forms a continuous and smooth curve. We aim for angular smoothness, as this makes it easy for the human eye to track corresponding edges that go through a node, linking the incoming edges to the corresponding outgoing edges in an intuitive fashion.

3.2 Techniques

We have designed a set of techniques that fulfill both design goals, while maintaining a representation that is close the classical graph-based layout. The techniques we have selected to build the smooth graph representation are as follows: 2D Graph Layout, Smooth N-th Order Interpolation, Edge Bundling, Interaction, Directionality Through Texturing. Each is discussed in more detail in the following sections.

3.2.1 2D Graph Layout

The states and transitions are visualized using a two dimensional graph layout. The main advantage of a two dimensional graph layout is that this allows for an intuitive representation of states (as nodes), while providing us the freedom to use curvature when drawing the edges. Also, the extra degrees of freedom introduced by using curved lines allow us to enforce smoothness and continuity constraints on the edges.

As a first step, we perform a classical 2D graph layout, based only on the first order transitions. From the existing graph layout algorithms in the literature, we have chosen to use the Fruchterman-Reingold force directed layout algorithm [4], a simple and robust algorithm that yields satisfactory results.

Once the layout algorithm is done, the locations of the nodes are set, but further refinements of the location may be made during the interaction process by the user. This allows the user to modify the proximity of the nodes so that visual groups can be built during the exploration process, as the knowledge the user has of the underlying phenomena grows.

3.2.2 Smooth N-th Order Interpolation

As stated in section 3.1.1, one goal is to visualize each unique higher-order transition individually. Since multiple higher-order transitions may contain the same first order transitions, there may be multiple edges between a pair of nodes that belong to different higher-order transitions.

A good vehicle for maintaining continuity and smoothness throughout all edges is to use interpolating splines. The user decides upon the transition length, N, and then we use the Catmull-Rom spline, so that each Nth-order transition can be represented by a spline of order N+1, as it should smoothly interpolate through N+1 nodes.

The advantage of this method is that the segments for successive transitions form one smoothly connected spline (see Figure 4), since the overlap between the higher-order transitions enforces smoothness along the connecting edges. This continuity constraint therefore guarantees that the entire state sequence, or any sub-sequence thereof, can be followed visually as a continuous line, greatly enhancing the visibility of the order in which states occur.

The only place where lines overlap, is where the higher-order transition is exactly the same, so where not only the from-and-to nodes of an edge are the same, but where the entire sequence of N+1 transitions is the same.

3.2.3 Edge Bundling

Since each higher-order transition maps to exactly one edge trajectory, we can bundle edges based on the frequency of the transition. We modulate the thickness of the edges to represent the number of times the transition was taken.

This is an implicit form of edge bundling, as all the edges taking part in a bundle are never explicitly drawn. Each unique state sequence, no matter how many times it occurs, is only drawn once, with thickness representing the number of occurrences. The thickness modulation makes it easy to track the frequently occurring transitions, and to find frequently occurring cycles within the data.

3.2.4 Interaction

To further inspect the state sequence, interactive selection can be used to investigate the graph. To assist the user in finding transitions of interest, we designed a user interface to incrementally build a constrained selection. The user can create such a selection by sequentially clicking on a number of nodes. All edges that connect the selected nodes in order then become part of the selection, and are visually highlighted (see Figure 7).

The selection process employs three steps. First, the starting state is selected by clicking. Then visual feedback is given by highlighting all possible continuations of the currently selected sequence, after which the selection criteria can be extended by selecting additional nodes.

By using color to highlight the selected edges, there is no need to make any changes to the layout of the nodes, which improves the visual coherence.

As we will demonstrate in section 5, the selection can be rapidly linked to correspond with all the ranges in time at which the selected sequence of transitions occurs.

As a secondary form of interaction, the nodes of the graph can be rearranged by the user. This creates direct feedback on which edges contain the moved node in their higher-order transition, as all these spline edges will interactively change shape when the node is moved.

Ware and Bobrow [22] have demonstrated that this class of dynamic highlighting, particularly when combined with other highlighting methods (static highlighting through coloration and motion highlighting using crawl edges, see section 3.2.5), is a useful aid for large graph querying.

3.2.5 Directionality Through Texturing

Classically, arrowheads are often used in graphs to visualize the direction of an edge. In our case, we want to be able to preserve the visual continuity that the edges have throughout nodes. Arrowheads distort the perception of the continuous spline, so we have opted for a basic-yet-effective approach of using texturing to visualize the orientation of the edges (see Figure 5).

Each edge is longitudinally split into two halves and each of these gets assigned a thickness, or width, based on the number of times that the transition in the corresponding direction occurs. A cyclically repeating texture is applied to each of the halves, and is animated in the direction of the state change that it represents in order to enhance its representation.

This is reminiscent of the approach presented by Wegenkittl et al. for the visualization of continuous state changes, called trajectories, through the four-dimensional state space of the Wonderland econometric model [23]. They employed spiral textures that were animated in the direction of the state change, mapped to the tube representing that trajectory.

Our texturing scheme proved flexible enough for later customization, so that further experiments can be done to find an optimal visual

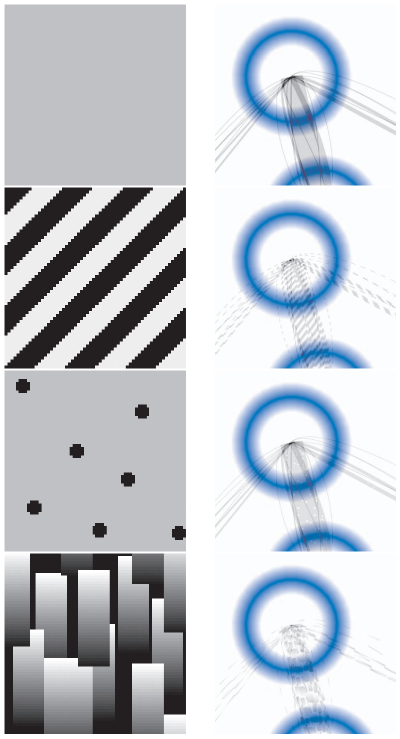

Fig. 5. Edges drawn with four different textures, from top to bottom: solid, striped, stippled and gradient. The left column shows the texture used, while the right column shows the resulting graph.

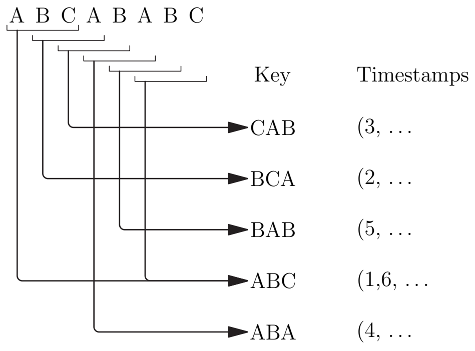

Fig. 6. Contents of the hash table after storing the sequence ABCABABC

representation for the directionality. As frequently used in flow visualization, we currently use an intensity gradient along the direction of the line to indicate orientation of the edges.

4 Implementation and Performance

We have implemented the proposed method as a functional component within a larger experimental data-exploration system. As the focus of the system lies on the exploration of large time-series data, we have required that all queries and drawing are performed at highly interactive frame-rates, even for larger data sets.

We use tuple hash tables to make searching the higher-order transitions efficient (see Figure 6). As the problem is similar to fixed-length string searching, a multitude of methods are available to efficiently count and enumerate the time-points at which a specific sequence of states occur. The hash table uses the Nth order transitions as keys, and maps those to a sorted array of time-points at which the corresponding sequence occurs.

These hash tables can not only be used to determine the width of each drawn edge (through counting the number of occurrences), but

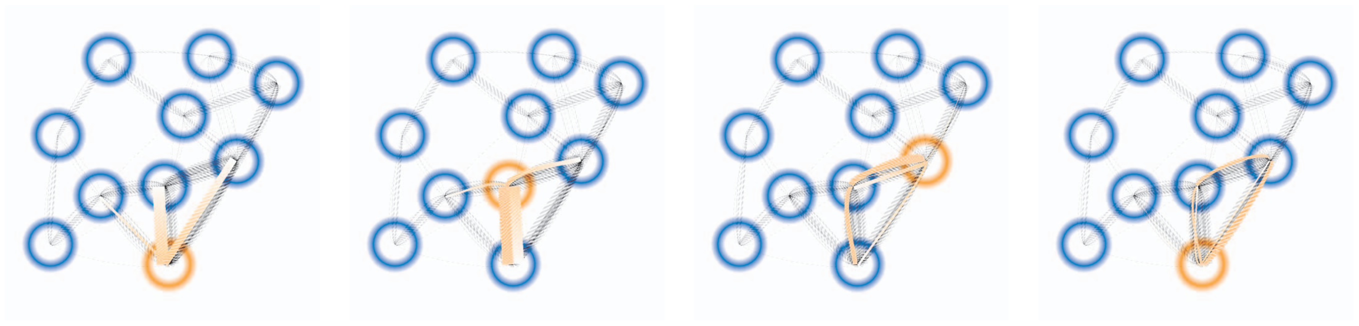
Fig. 7. The stages of the selection process, from left to right: A starting node is selected and three additional nodes are added to the sequence

Transition Order (N)	Edges (5 states)	Edges (10 states)
1	18	41
2	40	108
3	70	238
4	105	440
5	142	689
6	188	989

Table 1. The total number of edges in the graph for a real-world data-set of 1 million samples, as a function of the order of the transition. The original multi-scalar data was clustered with a chosen number of labels. The two columns show the results for 5 and 10 labels respectively.

they can also be used to perform the queries needed during the selection process. Once a selection of states has been made, the hash table can be queried to find all sequences that start with the selected sequence. When a limited range of time-points is to be examined, a binary search in the sorted time-point array suffices to filter the results.

Since each unique higher-order transition only has to be drawn once, the rendering is fast, even for large numbers of samples. Though the drawing of curved lines is a relatively slow operation, the number of edges drawn is sufficiently low that this does not impede overall performance.

In our data-set of one million data points and 10 unique labels, the number of 3rd order edges that had to be drawn was only 238. While in random state sequences the number of unique Nth order transitions scales exponentially with N, in natural data the number rises fairly slowly (see Table 1).

The drawing of both nodes and edges was implemented in OpenGL. The curved edges are drawn using strips of triangles, so that the thick curved edges can be filled efficiently. The total time it takes to draw a graph with 238 edges and 10 nodes is below 5 milliseconds on a modern workstation. This allows for the entire application to be used interactively. Figure 8 shows a snapshot of the rendering.

5 Application: Biological Sensor Data

The motivation behind the development of this visualization tool, as shown in Figure 9, was to enable researchers (biologists) to interactively explore biological sensor data. The multi-attribute data is collected from sensors attached to animal subjects over durations of hours to days at a frequency of 10–20Hz. This results in a large time-span (many hours/days) of small-scale behavior (tenths of seconds for scratching behavior to a couple of minutes for a diving behavior). The tags record data from an accelerometer, a magnetic compass and environmental sensors. Due to the large volume of data and complex inter-relationships at play between the different sensor channels the behavioral data is difficult to analyze, and requires extensive experience to interpret. Currently, tracking data is visualized as primitive, 2D time series plots (the ScalarView in figure 9), and analysis is based upon simple summary statistics. Interpreting and drawing conclusions about behavior from accelerometer data requires a great familiarity with both the behavior of the subject (to aid interpretation), and a great wealth of experience of working with accelerometer data of this form. Cognitive integration of experience with a scalar view of the data allow researchers to make statements about the behavior of the animal within its environment.

The cognitive cost can be reduced by employing clustering techniques upon the data, and visualizing the resulting cluster labels as part of the scalar view. This presents a good summary visualization, but for long time series data, it is still difficult to understand the behavior of the animal (for example by identifying the predominant behaviors or determining any unusual behaviors). The color bars above the scalar data in figure 9 give one example of visualizing the cluster information.

We have previously shown [6] that clustering and visualizing the data using spherical plots, spherical overlays, spherical histograms and a posture state graph are effective at leading to biological understanding. In that work, the biggest aid to simplifying a large data set is the *Posture State Graph* (PSG). For PSG visualization, each cluster is represented as a node in the graph, and then any transition between nodes is represented as an edge. Edge width can identify the number of transitions between nodes. Nodes correspond to particular orientation and behaviors and therefore present a visual summary of the data. In actual use, the most productive mode was to mix the spherical plot with overlay and PSG which gave good contextual information regarding predominant behaviors, leading to insight about behavioral sequences of the animal. The main problem with such a view was that it became cluttered. It presented a good overview, but still required some effective interaction to understand the relationship between predominant behaviors, unusual behaviors or to appreciate the time spent during particular behaviors.

The new smooth graph visualization directly addresses the problem of visual clutter by removing the need to have contextual information in the same window. Secondly through the use of curves, higher order transitions can be identified, which provides visual feedback of behavioral sequences, and edge bundling allows a simple appreciation of the frequency of those behavioral sequences, neither of which could be achieved without extensive interaction in previous visualization approaches.

The following sections introduce the multiple linked views of the exploratory framework and the different aspects of the data each presents. Some example explorations of the data will be given, with the insight the new approach provides.

5.1 Components

The screen is split in four viewports (see Figure 9): the GraphView, the SelectionView, the ScatterPlot and the ScalarView. The ScalarView displays the original scalar attributes measured at each time-point in a chart. This allows the user to see the scalar values that were used as a basis for the clustering step. As an overlay above this data, the states are visible as a colored bar. Colors are consistently chosen so that throughout all visualization the same color corresponds to the same cluster label. If a selection is active, for example for a specific sequence of states, then a secondary bar highlights all the ranges in the

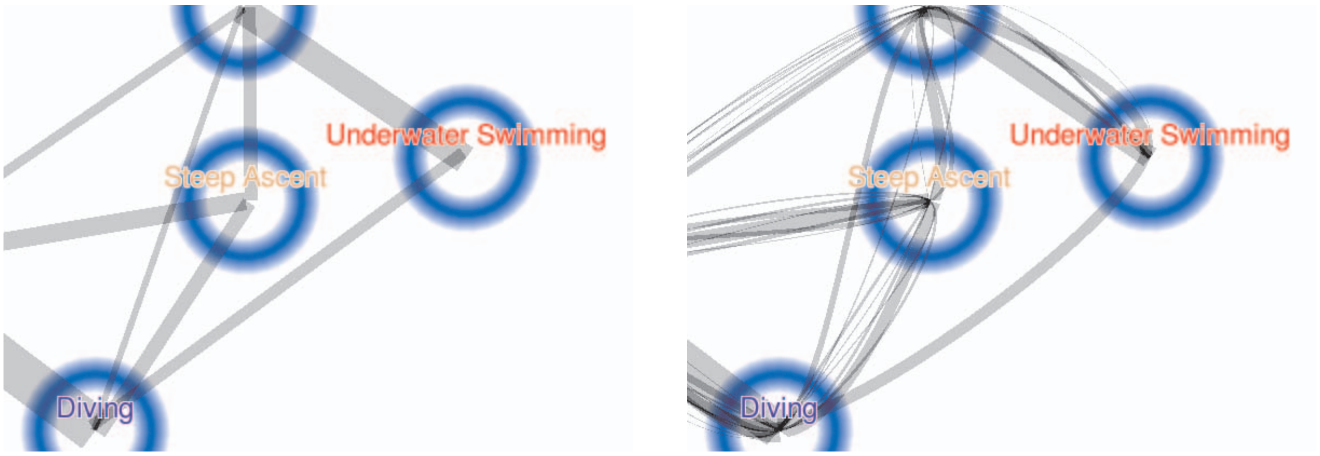

Fig. 8. Comparison between first-order (left) and third-order (right) state transition visualization. Notice how the edge between *diving* and *underwater swimming* consists of just a single line, even in the third-order representation. This directly indicates that this transition only occurs from within a single context, so there can be only a single specific node that precedes the transition, and there can be only a single specific node that succeeds it. Following the smooth edge in both directions, it is clear that the succeeding node is the top one, while the preceding node is apparently positions to the far left of the graph.

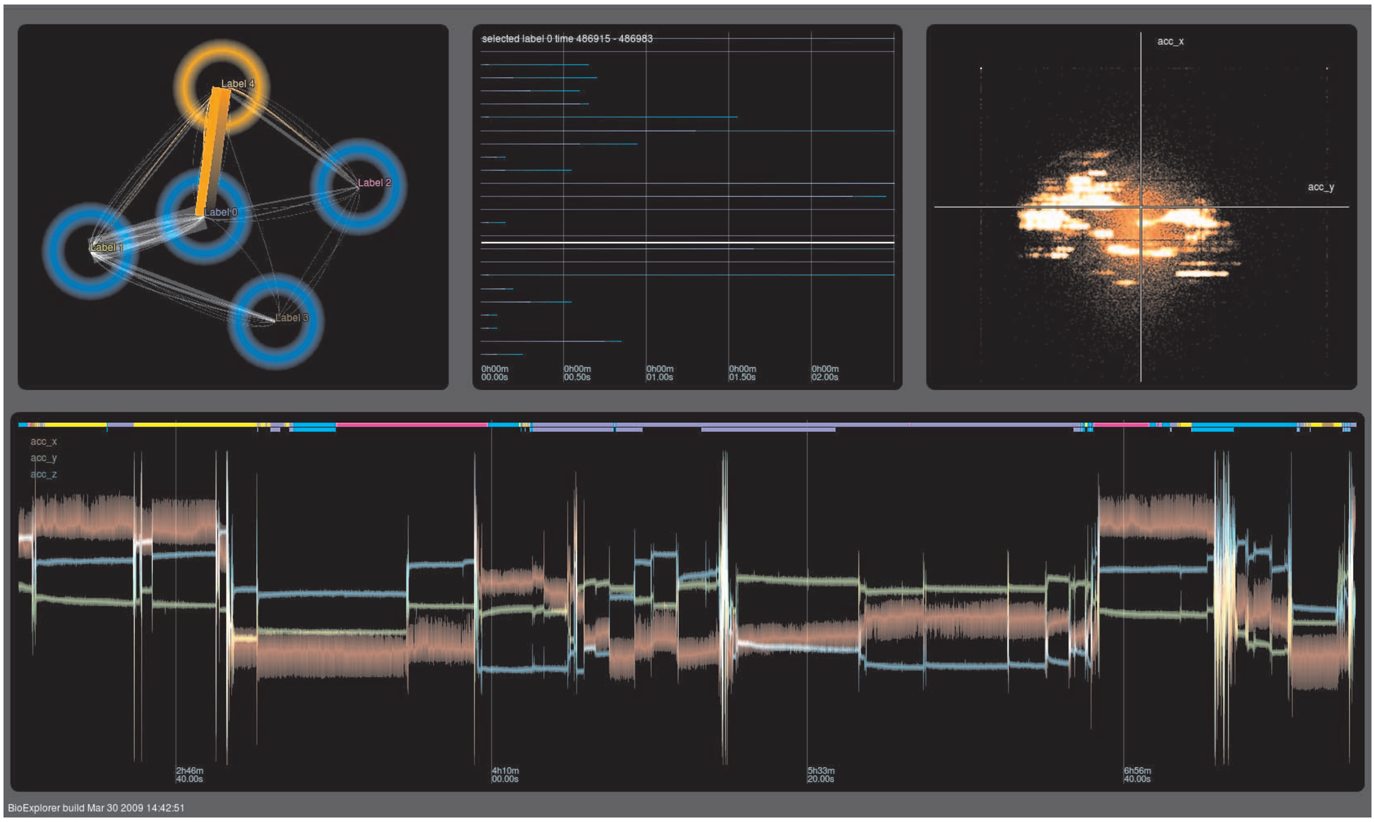

Fig. 9. A screenshot of the user interface of the exploratory application. The three views on the top of the screen represent (from left to right) the GraphView, the SelectionView and the ScatterPlot. The bottom of the screen shows a ScalarView in which both the accelerometer measurements as well as the assigned labels are shown.

time-series that match the current criteria. Since the number of data samples can be in the order of millions, the user can freely and quickly zoom in and out to focus on specific ranges in time, similar to [10].

The SelectionView gives an instant overview of the currently selected sequences. It depicts the time spent in each of the states within the selected sequences. Through this basic visual representation it provides the ability to make selections based on properties such as total duration of the sequence. This has been a valuable utility to spot and remove 'bad' labels that were introduced due to noise, as these often have a very short duration. The SelectionView also works as an aid to navigating the ScalarView, as when the displayed bars are clicked, the ScalarView will smoothly zoom to the corresponding time range.

The ScatterPlot displays a multi-attribute scatter-plot of the current selection, assisting in finding correlations between attributes. To cope with the large amount of data samples the points are rendered transparently, to keep the generated image clutter-free.

The GraphView links all the previously presented views by providing the main means of state space exploration. The previously described selection process is fully integrated into the linkage between the views, so that during the construction of a selection set, all views correspond to the data associated with the selection. To cope with the limited amount of screen space available in this split-screen scenario, the graph view supports both zooming and panning, so the area of interest can be changed interactively.

5.2 Exploration

To examine the performance of our method on real-world data, we have used the exploratory framework to analyze a data-log containing sensor information from a penguin. Over the entire course of the log, several different activities occur, such as diving and swimming. The data is labelled using a k-means clustering algorithm which assigns labels based on the posture information extracted from the accelerometer data.

5.2.1 Finding Repeating Patterns

When exploring the penguin data, we are interested in finding the main activities, and then reasoning about any activities that follow a different pattern. The clustering identifies the various states of the animal, and when related to body posture these can be labelled as seen in figure 10 (diving, underwater swimming, ascent, steep ascent and surface swimming). The edges show where transitions occur between states, edge thickness shows the frequency of those transitions, and edge continuity shows behavioral sequences within the data.

Over and above the visual information provided by previous techniques, researchers can now see a summary of the behavioral sequences and can begin to reason about the animal from just a static view of the data. For instance we can see a continuous path going from diving, through underwater swimming, ascent, steep ascent, and back to diving. This shows the main cycle of a feeding penguin (we shall call this a dive sequence in the following). By selecting that path (figure 11) we are then presented with further statistical summary in the form of the SelectionView which shows us that the dive sequence lasts about 15–20 seconds in the majority of cases. By selecting one of the dive sequences in the SelectionView, the ScalarView shows us the original scalar data where the biologist can verify that this labelling is correct through the pattern known from experience.

In fact the full dive sequence involves another selection of "diving" before the penguin enters the surface swimming state. This is due to the classifier interpreting the body orientation the penguin adopts as it exits the water (to avoid its inherent buoyancy from projecting it out of the water), being similar to that when it prepares to dive.

During development we found it useful to include the ability to sort the SelectionView by the total duration of the sequence. In this case it clearly shows the expected normal distribution for dive length.

From the GraphView we can see another thick path between Surface Swimming and Diving. This is the penguin performing shallow dives in order to swim efficiently just below the surface, before surfacing to breathe, and then diving again. The SelectionView shows that each lasts about 5 seconds. Sometimes it dives a little bit deeper, and

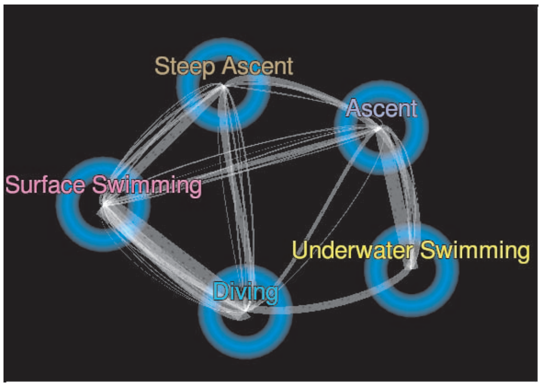

Fig. 10. Smooth graph of 5 states in penguin data set. The data has been classified into the states shown using k-means clustering. The label colors correspond to the color bar in the ScalarView and the individual selection bars as seen in Figure 11.

goes through the ascent and steep ascent transitions (but this is a rare occurrence in the data as conveyed by the thinner edge transitions).

5.2.2 Noisy Data

Another advantage of the higher-order visualization is that the edges that move back-and-forth between two nodes repeatedly are separated from the edges that make the transition as part of a complex higher order transition. For example, the thick straight edge between *Surface Swimming* and *Steep Ascent* indicates that many transitions occur between these two nodes. We can examine this phenomenon closer by building a selection from *Surface Swimming* to *Steep Ascent* and back again. Through examining the SelectionView, we learn that the duration of these two states is often very short, as the colored bars are fairly short. When we select one of the bars, the scalar view zooms in on the time range of the occurrence and we can establish why there are so many back-and-forth transitions. Looking at the scalar values, we see no particular patterns that warrant these erratic state changes, as indicated by the label bar on top of the figure.

The logical explanation for this is that the boundary between these two clusters is particularly sensitive to noise, therefore values that are in between the two clusters are often misclassified. As our sensor data is inherently noisy, this particular group of transitions should be refined before further analysis. Another course of action can be to decide that the two states probably refer to a very similar posture, and that they should therefore be merged into a single state. We can visually perform this action by positioning the two nodes on top of each other, or we can reprocess the label data and substitute one state for the other to get rid of the state entirely.

6 Conclusions and Future Work

We have presented a novel visual representation for state transition graphs that focuses on the temporal coherence between sequences of transitions. Our representation uniquely visualizes higher-order state transitions using a smooth spline representation for the edges. The method we designed adheres closely to the principles of continuity, connectedness and closure.

The higher order transitions are represented as curved lines in such a way that each sequence of states forms a smooth continuous curve through their respective nodes. The visual continuity of the curve is easily tracked by the human eye, making correlations between ingoing edges and out-going edges visible in an intuitive way. The continuity holds for state sequences of arbitrary length.

The fact that the presented method uniquely represents each Nth-order state transition visually has proven useful in finding frequently occurring state transitions in noisy real world data. We have demonstrated this and other issues by exploring a real-world dataset.

The limitations of our method manifest when there are either a large number of states to visualize, or when even higher order state transi-

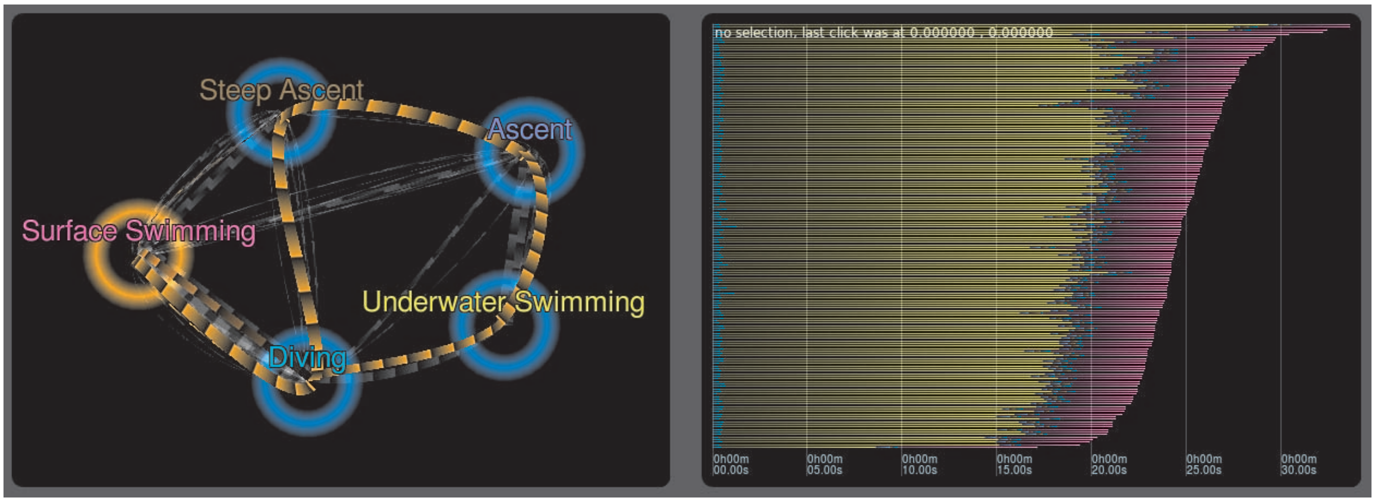

Fig. 11. Selection of the penguin's feeding cycle by querying the sequence of nodes corresponding to diving, underwater swimming, ascent, steep ascent, diving and surface swimming.

tions need to be visualized. In both cases the visual complexity increases greatly, complicating the use of our method. In the former case, the number of nodes will serve to clutter the display and leave little room for routing the edges. In the latter case, the higher order of transitions simply leads to more permutations of the spline interpolation, or more types of edges. This can also be seen in table 1, where increasing the order of the transitions leads to an exponential increase in the number of edges. However, for the datasets we have experimented with up to now, neither of these limitations has posed any problems.

In future work, we are planning to investigate whether the idea of representing higher order state transitions using curved edges can be successfully applied to other state transition graph visualization methods in addition to two dimensional node-link diagrams.

We are also planning to continue our investigation into the visualization of oriented edges by using animated texture, taking into account the recent work by Holten and van Wijk [9].

Although tested through investigation with a biological sensor dataset, it is also important to evaluate the performance of curved edges when employed for typical tasks. We are planning a user study of the type carried out by Ware and Bobrow [22], and Holten and van Wijk [9]. In particular we flag that the choice of an appropriate basis for comparison will require a careful experimental design.

Acknowledgments

We would like to thank Rory Wilson of the Institute of Environmental Sustainability (IES) at Swansea University for the penguin data set. This research is supported by the Netherlands Organisation for Scientific Research (NWO), project number 643.100.503 MFMV.

References

[1] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, Nov.-Dec. 2008.

[2] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent layered drawings. *Algorithmica*, 47(4):439–452, 2007.

[3] D. J. Field, A. Hayes, and R. F. Hess. Contour integration by the human visual system: Evidence for a local "association field". *Vision Research*, 33(2):173–193, 1993.

[4] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.

[5] M. Graham and J. Kennedy. Using curves to enhance parallel coordinate visualisations. In *Proc. Information Visualization*, pages 10–16, 2003.

[6] E. Grundy, M. W. Jones, R. S. Laramee, R. P. Wilson, and E. F. Shepard. Visualization of sensor data from animal movement. *Computer Graphics Forum (Eurovis)*, 28(2):815–822, 2009.

[7] I. Herman, G. Melancon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, Jan-Mar 2000.

[8] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sept.-Oct. 2006.

[9] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. In *Proc. CHI '09*, pages 2299–2308, 2009.

[10] D. Jerding and J. Stasko. The information mural: a technique for displaying and navigating large information spaces. *Visualization and Computer Graphics, IEEE Transactions on*, 4(3):257–271, Jul-Sep 1998.

[11] T. Jéron and C. Jard. 3d layout of reachability graphs of communicating processes. In *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing*, pages 25–32, 1995.

[12] M. Leuschel and E. Turner. Visualizing larger state spaces in ProB. In *Proceedings of the International Conference of B and Z Users*, 2005.

[13] A. Pretorius. *Visualization of State Transition Graphs*. PhD thesis, Eindhoven University of Technology, 2008.

[14] A. J. Pretorius and J. J. van Wijk. Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):685–692, 2006.

[15] A. J. Pretorius and J. J. van Wijk. Visual inspection of multivariate graphs. *Computer Graphics Forum*, 27(3):967–974, May 2008.

[16] H. Theisel. Higher order parallel coordinates. In B. Girod, G. Greiner, H. Niemann, and H. Seidel, editors, *Proc. Vision, Modeling and Visualization 2000*, pages 119–125, Saarbrücken, 2000.

[17] F. van Ham and B. Rogowitz. Perceptual organization in user-generated graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1333–1339, Nov.-Dec. 2008.

[18] F. van Ham, H. van de Wetering, and J. J. van Wijk. Visualization of state transition graphs. In *Proc. IEEE Information Visualization*, pages 59–66, 2001.

[19] F. van Ham, H. van de Wetering, and J. J. van Wijk. Interactive visualization of state transition systems. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):319–329, 2002.

[20] C. Ware. *Information Visualization: Perception for Design (Morgan Kaufmann Interactive Technologies Series)*. Morgan Kaufmann Publishers, 1st edition, 2004.

[21] C. Ware, R. Arsenault, M. Plumlee, and D. Wiley. Visualizing the underwater behavior of humpback whales. *IEEE Computer Graphics and Applications*, 26(4):14–18, July-Aug. 2006.

[22] C. Ware and R. Bobrow. Motion to support rapid interactive queries on node–link diagrams. *ACM Transactions on Applied Perception*, 1(1):3–18, 2004.

[23] R. Wegenkittl, M. E. Gröller, and W. Purgathofer. A Guided Tour to Wonderland: Visualizing the Slow-Fast Dynamics of an Analytical Dynamical System, Apr. 1996. human contact: technical-report@cg.tuwien.ac.at.