Interactive Visualization of State Transition Systems

Frank van Ham, Huub van de Wetering, and Jarke J. van Wijk

Abstract—A new method for the visualization of state transition systems is presented. Visual information is reduced by clustering nodes, forming a tree structure of related clusters. This structure is visualized in three dimensions with concepts from cone trees and emphasis on symmetry. A number of interactive options are provided as well, allowing the user to superimpose detail information on this tree structure. The resulting visualization enables the user to relate features in the visualization of the state transition graph to semantic concepts in the corresponding process and vice versa.

Index Terms—Graph visualization, transition systems, state spaces, cone trees.

1 Introduction

The last decade has seen substantial progress in the analysis of computer-based systems. A central approach in this analysis is to describe the behavior by means of a transition system [1] consisting of states and possible transitions between these states. The most commonly used techniques based on explicit state enumeration can now deal with transition systems of billions of states. So-called symbolic techniques deal with even larger systems, employing a form of abstracted state spaces. These techniques have become sufficiently effective to assist in the design of systems and are typically and increasingly often used for advanced bug detection.

These advances in technology confront us with state transition graphs of enormous dimensions. Very little insight into the structure of transition systems reflecting the behavior of computer-controlled systems exists, whereas such insight would be helpful in improving upon the reliable construction of such systems. Currently, three main approaches for the analysis of large transition systems exist. The first classical approach is to derive a much smaller abstract variant of the transition system (typically less than 100 states) and perform analysis on that. A disadvantage here is that information about the actual state space is lost, though this information could be very useful for testing purposes. A second approach consists of asking specific questions or checking for specific conditions. Although this works well for the detection of deadlock states, properties like symmetries within or similarities between processes can be nearly impossible for a computer to find. Another disadvantage is that asking specific questions requires specific knowledge about the system, which may not be available. As a third alternative method, we opt to use interactive visualization methods as the human perceptual system is much more adept at finding similarities between structures.

Visualizations are straightforward when dealing with a small number of states since a simple node and link diagram can be used. In practice, the finite state machines under analysis often consist of tens of thousands of nodes or more, which severely limits the usefulness of the traditional node link diagram (Fig. 1). Here, we present a new method to visualize large state transaction systems. The user is presented with an overview of the structure of the graph and can view individual nodes and edges on demand. Symmetries and regular structures in the graph are maintained and clearly visualized as such. Various properties of the state space parameters can be visualized interactively on top of this structure.

Section 2 discusses related work and the general ideas behind our method. Section 3 describes the method used to visualize the state transition graph in more detail, including an efficient algorithm to cluster the nodes. In Section 4, we present a variety of methods to obtain additional insight in the state transition system. Attributes of the states are visualized using color and shape; various options for interactive queries and selection are presented. In Section 5, we discuss a number of real world cases, with up to a million nodes. Finally, in Section 6, conclusions are drawn.

2 Approach

Much research has been done in the area of two dimensional (2D) graph drawing. For an overview, see [2]. The general approach for 2D visualizations of directed graphs is to produce a pleasant looking picture by optimizing one or two aspects of the visualization. In the case of larger graphs, however, the lack of visualization space in 2D quickly becomes a problem. A second problem is that optimizing only one aspect loses its effectiveness when dealing with a large number of nodes. Several attempts to overcome these problems have been made by using 3D visualizations and by applying techniques such as hyperbolic space [11], hierarchical node clustering [15], [4], or self-organizing

• *The authors are with the Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, PO Box 513, 5600 MB Eindhoven, The Netherlands.*
E-mail: {fvham, wstahw, vanwijk}@win.tue.nl.

Manuscript received 15 Feb. 2002; revised 15 Mar. 2002; accepted 2 Apr. 2002.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 116209.

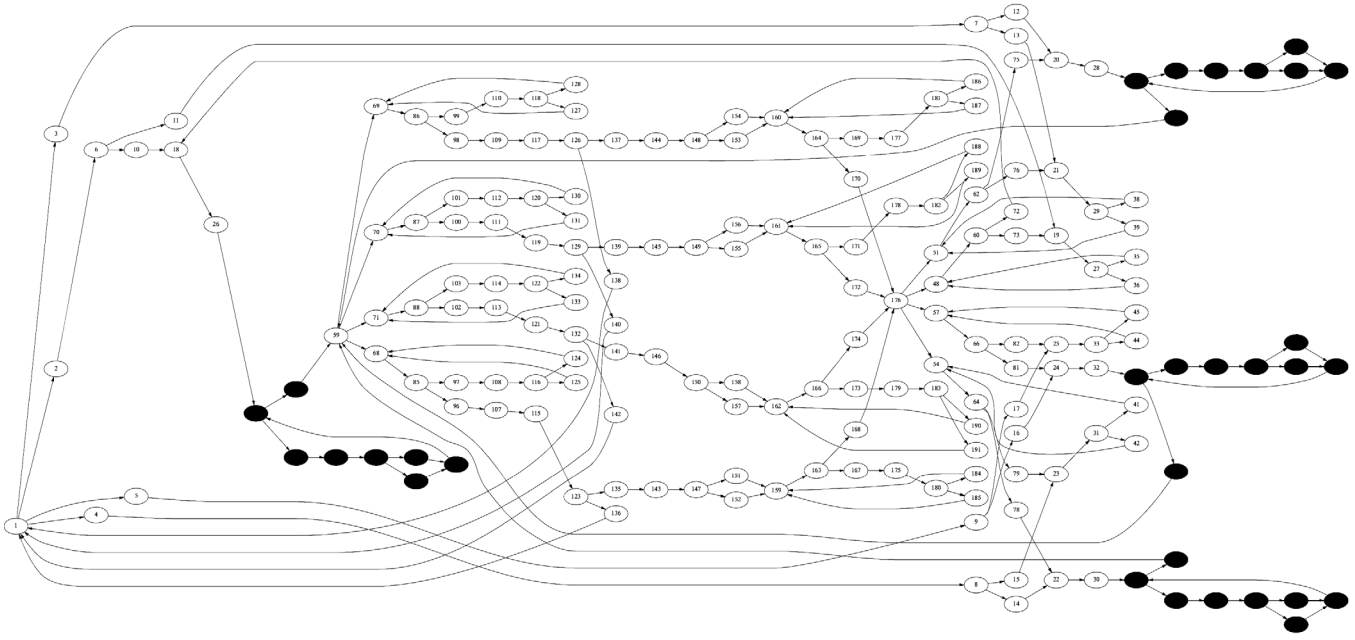

Fig. 1. Two-dimensional visualization of a state transition graph with 191 states.

networks [8]. Since finding a general solution for visualizing large directed graphs can be difficult, most existing methods are forced to use domain-specific information. In this case, we focus on large state transition graphs, which are automatically generated from high-level process descriptions and generally exhibit some form of symmetry and regularity. The visualization method presented in this article relies on three principles:

1. **Enable the user to identify symmetrical and similar substructures:**
   By choosing to optimize only one local aspect of the graph, such as edge crossings, many structural symmetries or similarities are ignored. The same goes for extracting a minimal weight spanning tree. The visualization in Fig. 1, for example, obscures the fact that the marked groups of nodes have identical structural properties by positioning them in different ways. A clear picture of the similarities (and, to a lesser extent, the symmetries) in a graph enables users to mentally break up large structures into smaller, similar-looking pieces, thus making a complex picture easier to digest. It also facilitates analysis and comparison of similar structures.
2. **Provide the user with an overview of the global structure of the entire graph.**
   A major problem in the visualization of state transition graphs is the sheer size of the graph. No matter the quality of the layout, it is simply impossible for the human perceptual system to form a schematic idea of what a graph looks like when confronted with thousands of information elements. A technique to drastically reduce visual complexity is the grouping of nodes based on a common property, also known as clustering [9]. Clustering based on the structure of the graph is especially useful here since the organization of the resulting clusters generally resembles the overall structure of the original graph, which facilitates maintaining context. Structure-based clustering can provide a useful high-level map of the graph in which the user can then select an area of interest to be inspected more closely.
3. **Provide sufficient interaction such that user questions about various aspects of the system can be answered.**
   Since not all information present in the original system is taken into account when constructing the visualization, we have to provide the user with other mechanisms to obtain this information. He has to be enabled to examine the system from multiple perspectives, focus on parts of the visualization he is interested in, and obtain detail information on demand. Interaction plays a crucial role in this.

3 Visualization of the Graph

Assuming we have a Finite State Machine consisting of a finite set of states and an also finite set of possible transitions (along with conditions) between these states, we can define the corresponding state transition graph by the graph $G = (V, E)$, where $x \in V$ represents a state and $a_{xy} \in E$ represents a *directed* edge (or *arc*) between the nodes x and y. A start node $s \in V$ represents the finite state machine's initial state. We split the visualization process for a state transition graph into four distinct steps:

1. Assign a rank (or layer) to all nodes.
2. Cluster the graph based on a structural property, resulting in a backbone tree structure for the entire directed graph.
3. Visualize this structure using a method related to cone trees.
4. Position and visualize individual nodes and edges.

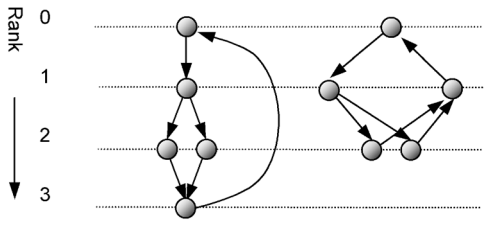

Fig. 2. Different views on the same process: (a) iterative and (b) cyclic.

3.1 Ranking

In a first step, nodes are assigned to a layer or rank, comparable to the standard Sugiyama-type layout [13]. That is, all nodes are assigned a nonnegative discrete value that indicates their depth relative to the start node. We found two ranking methods to be the most useful since they correspond to two types of views on processes: iterative and cyclic (Fig. 2).

In an *iterative* process view, the start node s is assigned rank 0 and subsequent nodes are assigned a rank that is equal to the length of the shortest path from the start node to that specific node, similar to a breadth first search (see Fig. 2a). This ranking has the advantage that most arcs between ranks point in the same direction, which creates a natural "flow" in the picture. Since most people tend to think of processes in terms of an iterative execution of statements, users will likely be familiar with this way of visualizing processes. The biggest disadvantage, however, is that the few arcs that do point in the other direction are usually longer. These *backpointers* may not present a significant problem when dealing with relatively simple processes, but, when dealing with complicated processes, they tend to span several ranks and spoil the final visualization.

In a *cyclic* process view, we view a process as a cyclic execution of statements, with no clear beginning and end. If the start node is assigned a rank of 0, other nodes are assigned a rank that is equal to the length of the shortest path from that node to the start node, *independent of the direction of the edges* (see Fig. 2b). This type of ranking by definition eliminates any long backpointers in the visualization since each node is positioned at most one rank away from any connected node. This may be advantageous if users are looking for connected clusters of nodes. The major disadvantage is that arcs between ranks do not point in one direction, which makes it harder to comprehend the graphs layout in detail.

From here on, we refer to arcs that point from a low ranked node to a higher ranked node as arcs pointing down and vice versa. Furthermore, note that horizontal arcs connecting nodes in equal ranks can also occur.

3.2 Clustering Process

In the second step, nodes are clustered to reduce the visual complexity of a graph, based on a local structural property that we define in this section. Instead of clustering by focusing on a single global property, such as "minimal edges between resulting clusters" or a property of a single node, we propose a clustering based on an equivalence relation between nodes.

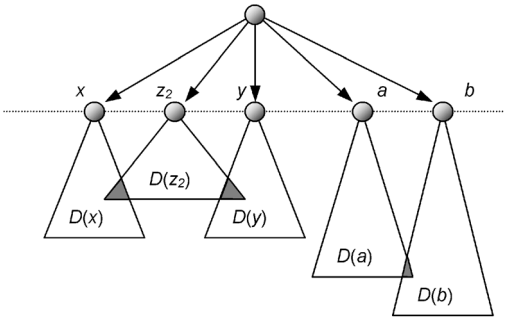

Fig. 3. Nodes *x* and *y* are equivalent, as are nodes *a* and *b*, while nodes *x* and *b* are not.

We aim at the creation of a tree structure based on the original graph, which we can use as a backbone to display the complete structure of the entire graph. To this end, we first modify the original simple directed graph and, next, simplify this modified graph by clustering the nodes.

Given a graph G, consisting of a set of nodes V and a set of arcs E and given a ranking R that maps nodes to a nonnegative discrete rank, we define a new set of arcs E' by removing any arcs spanning more than one rank from E. Additionally, to facilitate definitions, we reverse the direction of the remaining arcs that point upward. More formally:

$$E' = \{a_{xy} \mid x, y \in V \wedge a_{xy} \in E \wedge 0 \leq R(y) - R(x) \leq 1\} \cup \{a_{yx} \mid x, y \in V \wedge a_{xy} \in E \wedge R(x) - R(y) = 1\}.$$

No arcs in the graph $G' = (V, E')$ are pointing upward since, for all a_{xy} in E', $0 \leq R(y) - R(x) \leq 1$ holds. Note that this does not necessarily mean that G' is acyclic since a cycle consisting of nodes all having equal rank is still possible. Let $D(x)$ be the set of all nodes that can be reached from x via zero or more arcs in E'. We now define two nodes x and y to be equivalent iff a row $(x = z_1, z_2, \ldots, z_N = y)$ of nodes with equal rank exists such that for all $1 \leq i < ND(z_i) \cap D(z_{i+1})$ is not empty (see Fig. 3).

It can be shown [6] that the relation outlined above is an equivalence relation, so, by definition, its equivalence classes are nonempty and disjoint, which makes them very suitable to use as clusters. Since all nodes in a cluster have the same rank, we can extend the concept of rank to clusters. The rank of a cluster containing node x is then equal to the rank of x. We can now define a relationship between clusters that can be used to construct the backbone structure: A cluster C_1 is defined to be an *ancestor* of a cluster C_2 iff $Rank(C_1) = Rank(C_2) - 1$ and there exists an arc in E' connecting a node in C_1 with a node in C_2. A cluster C_2 is defined to be a descendant of C_1 iff C_1 is an ancestor of C_2. Since each node is in exactly one cluster, each cluster has at most one ancestor and cyclic ancestor relations are not possible, we can state that the resulting cluster structure is a tree (Fig. 4).

3.3 Clustering Algorithm

In a usable application, the implementation of this clustering process will have to be linear. Instead of using the naive approach of storing $D(x)$ for each node x and checking for a nonempty intersection, we devised a recursive algorithm

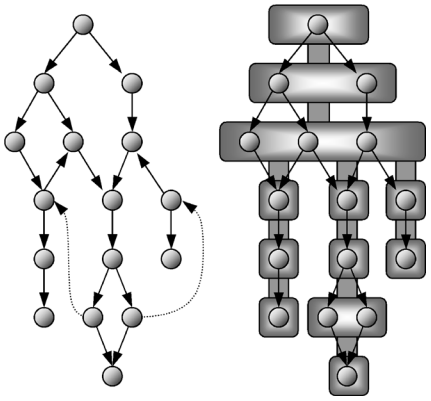

Fig. 4. Original graph G (left) and clustered graph G' with backbone tree (right).

that clusters all nodes in linear time. In this paragraph, we outline this algorithm for computing the backbone tree of a graph $G = (V, E)$, where each node n has been given a rank $R(n)$. For this, we consider the graph G', as defined before.

Consider the nodes v and x with a_{vx} an arc in G'. According to the definition of G', there are two cases for x, either $R(x) = R(v)$ or $R(x) = R(v) + 1$. In the first case, node x is equivalent to node v since $D(x)$ and $D(v)$ have a nonempty intersection containing at least x. So, x and v are in the same cluster.

The second case is more complicated. If the ranks of v and x differ, they are not in the same cluster. However, arcs crossing ranks can induce that v has to be merged with other nodes without a direct connection with v. We therefore first compute the cluster of x. For all nodes y in the cluster of x, we can state that $D(x) \cap D(y)$ is not empty. Hence, for all these nodes y, we have to add all nodes w with an arc $a_{wy} \in E'$ and a rank equal to $R(v)$ to the cluster of v since $D(v) \cap D(w)$ is not empty (see Fig. 5).

These two cases form the heart of the recursive procedure `ClusterTree(v:Node,c:ClusterNode)` given below. The precondition of this procedure is that all nodes in `c.Nodes` are in the same cluster as node `v` and `v.Cluster` = **nil**, that is, `v` has not yet been assigned to a cluster. A postcondition for the procedure is that the subtree of the backbone tree with its root at the cluster of `v` has been fully computed.

```
type ClusterNode =
  record
    Nodes : set of Node;
    Anc : ClusterNode;
    Desc : set of ClusterNode;
  end;

procedure ClusterTree(v:Node,c:ClusterNode)
begin
  c.Nodes := c.Nodes ∪ {v};
  v.Cluster := c;
  forall a_vx in E' with x.Cluster=nil do
    if R(x)=R(v) then ClusterTree(x,c);
    if R(x)=R(v)+1 then
      ClusterTree(x,new ClusterNode);
      x.Cluster.Anc := c;
```

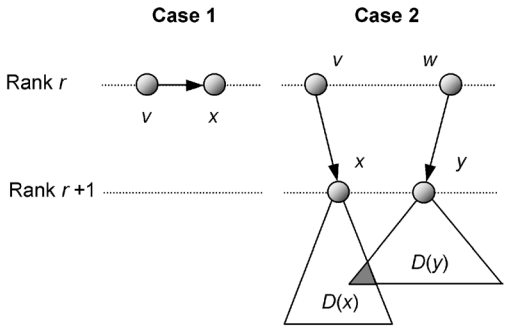

Fig. 5. Cluster algorithm case analysis.

```
    c.Desc := c.Desc ∪ {x.Cluster};
    forall y in x.Cluster.Nodes do
      forall a_wy in E' with w.Cluster=nil do
        if R(w)=R(v) then ClusterTree(w,c);
end;
ClusterTree(StartNode, new ClusterNode);
```

Listing 1. Clustering algorithm

The cluster algorithm has a time complexity linear in both the number of edges and the number of vertices of G' since each edge is traveled at most once and each vertex is a parameter for the procedure at most once.

3.4 Visualizing the Backbone Tree

Before making a choice for a layout, we first state our requirements for a good layout:

- **Symmetry is important** and, therefore, a visualization that produces a more symmetrical picture is to be favored. Clusters and nodes with the same structural properties should be treated in the same way.
- **There has to be a clear visual relationship between the backbone structure and the actual graph**. It is easier for the user to maintain context when inspecting a small detail section if this detail looks approximately the same in close up view as it did in the global overview.
- The size of the clusters has to be related to the number of nodes in a cluster to prevent cluttering. **Clusters with a larger number of nodes have to be visualized by larger visual elements.**

Although classical 2D layouts are very predictable, familiar, and easy to use, the lack of visualization space quickly becomes a problem when dealing with larger graphs, especially when considering that we want to visualize larger clusters as larger nodes in the tree. A popular technique to deal with this problem is to move from a 2D to a 3D layout, which gives us an extra dimension to increase the cluster size.

We aim for a visualization that depicts clusters as circles in a horizontal plane. A plane is reserved for each rank, with the topmost plane containing clusters with rank 0. The backbone tree is laid out in a manner resembling cone trees [12], with ancestor clusters positioned at the apex of an imaginary cone and their descendant clusters placed at the base of the cone. In other words, our clusters have the same

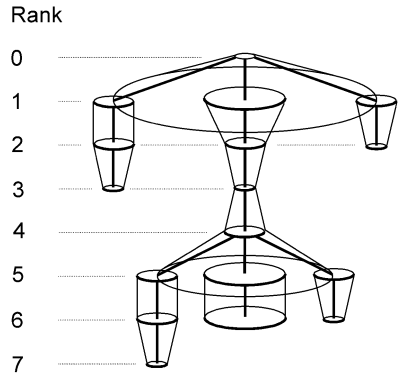

Fig. 6. Sample visualization.

status as tree nodes in a cone tree. To emphasize the hierarchy in the cluster structure, truncated cones are drawn between related clusters. Fig. 6 gives a quick impression. The overall process adheres to the basic concepts of cone trees, but with a few alterations:

1. Clusters (the nodes in the cone tree) are visualized as circles of different sizes.
2. Symmetry is improved by also allowing clusters to be positioned in the center of the cone's base.
3. The final resulting structure is given more "body" and some extra visual cues are added.

Ad A. Normal cone trees consist of a collection of similar looking nodes. The tree nodes in our modified cone tree, however, are the clusters we defined in the previous section. Since each cluster contains a different number of nodes, we represent them by different sized circles. Nodes will be placed on the circle boundary, so we choose to keep the circle's circumference proportional to the number of nodes in the cluster, which results in the same amount of visualization space for each node.

Ad B. We present a heuristic for creating symmetrical layouts and discern the following cases for the positioning of the N descendant clusters of a cluster A.

If $N = 1$, the descendant cluster is positioned directly below A. If $N > 1$, we space the clusters evenly over the base of a cone, with its apex at the center of A. The base diameter of this cone can be computed by using a recursive method similar to the one used by [3]. However, since positioning all N descendant clusters over the base may not always yield a symmetrical solution, we make the following three exceptions:

- If there is a unique largest cluster among the descendant clusters, we position this cluster directly below A in the center of the cone's base (Fig. 7a).
- If there is one unique smallest cluster among the descendant clusters, we center this cluster when there are no largest clusters centered (Fig. 7b) or when there is a largest cluster centered and the smallest cluster does not have any descendants. This prevents clusters from potentially overlapping each other.
- If, after centering clusters based on the above exceptions, only one noncentered cluster remains, we choose not to center the largest cluster. This produces a more balanced layout (Fig. 7c).

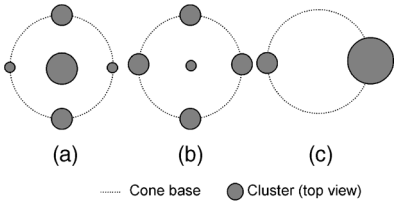

Fig. 7. Layout of individual clusters.

Ad C. Since nodes are positioned on the circle boundaries, most edges between nodes in a cluster and nodes in a descendant cluster will typically run within a section of space bounded by a truncated cone. A simple but effective way to reduce the visual complexity of the graph then is to visualize these two clusters as a truncated cone. The cone's top radius is set equal to the radius of the ancestor cluster and the cone's bottom radius is equal to the radius of the descendant cluster. If we are dealing with multiple descendant clusters, the cone's bottom radius is equal to the radius of the base of the (imaginary) cone the clusters are positioned on. Although this method provides a good overview of the graph's global structural properties, it suffers from some problems inherent to 3D visualizations, the most notable being the problem of objects occluding each other. To overcome this problem and at the same time improve use of available visualization space, we rotate noncentered clusters (and their descendants) slightly outward. Finally, transparency is added, which further reduces this effect and provides extra visual clues when looking at still pictures. Some examples are shown in Sections 4 and 5.

3.5 Positioning Individual Nodes

The previous two paragraphs presented a method to reduce visual detail by clustering nodes, providing a better global overview. The next step is to assign an optimal position to the individual nodes in the graph, given the fact that nodes are positioned on the circle edge. An optimal positioning of nodes satisfies the following requirements:

1. Short edges between nodes. A visualization is more effective if a node is kept close to its neighbors.
2. Maximum possible distance between nodes in the same cluster. Nodes are to be kept as far apart as possible to reduce cluttering and may not coincide.
3. Where possible, emphasize symmetry in the structure by positioning nodes with the same properties in the same way.

Clearly, the first two requirements contradict since positioning two nodes in the same cluster that have the same parent node further apart leads to a greater total edge length. Another problem is the computational complexity. Although positions can be calculated by minimizing an error function or using a force directed approach, the number of nodes we are dealing with is generally too large to provide visualization at an interactive level. Another disadvantage is that two runs of the same optimization algorithm on virtually identical graphs may produce radically different layouts, which makes it impossible to

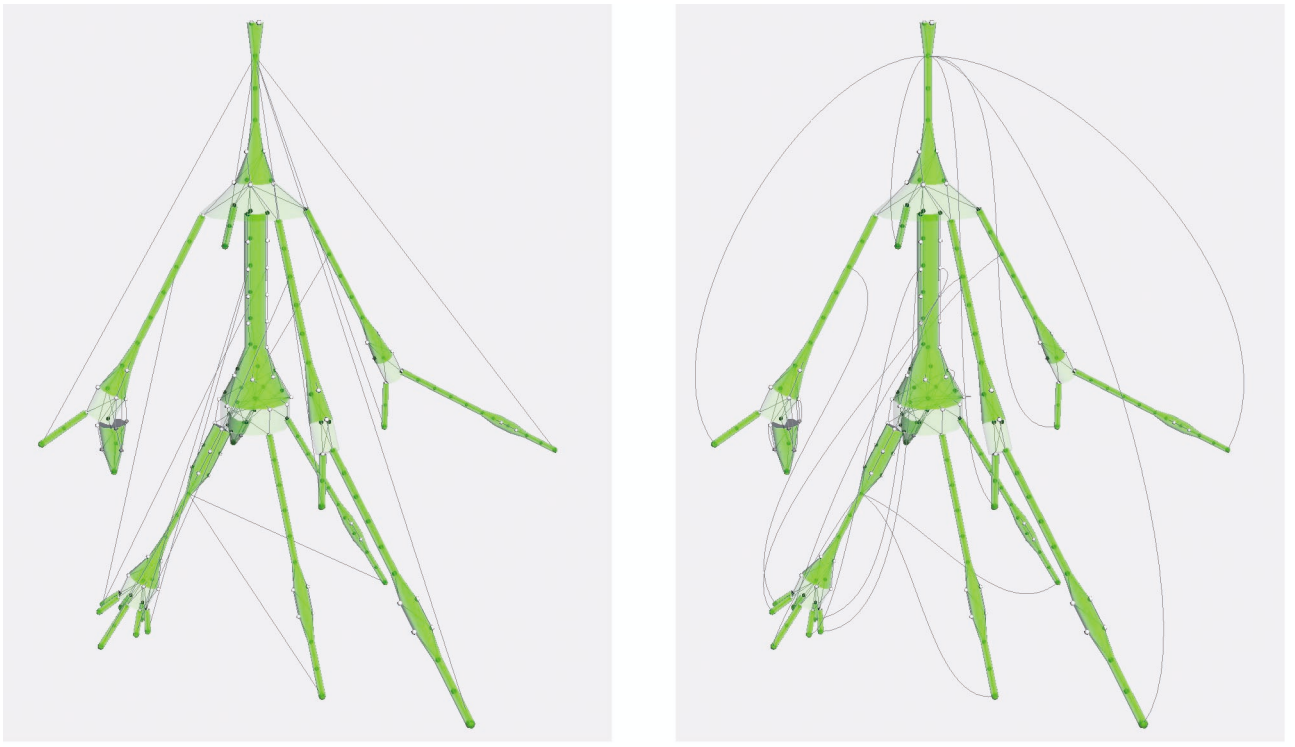

Fig. 8. Visualizing backpointers: (a) straight and (b) curved.

identify similar substructures within a graph. We therefore select a rule-based approach in which the position of a node is governed by local structural node properties.

We use a two-step heuristic to position the nodes. First, we assign initial positions, based on the positions of nodes in ancestor clusters, similar to [14]. That is, nodes are positioned at the barycenter of their parents position. To enforce some regularity on the resulting layout, each cluster is subdivided into a number of slots, after which nodes are rounded to the nearest slot. In a second step, we adjust these positions to increase the internode distance by dividing nodes sharing a slot symmetrically over a section of the cluster. The size of this section is governed by the occupancy of neighboring slots. A more detailed description of the layout method can be found in [6].

3.6 Visualization of Edges

Given the positions of the nodes, the edges between them can be visualized. The standard way to show edges is simply to draw a straight line between two nodes. Edge direction is then usually depicted with a small arrowhead or with transparency or edge thickness. We found that such subtle cues are not effective here because of the huge amount of edges. Also, the use of color is not the most intuitive, preattentive cue. How can we show direction more effectively? Inspired by Fig. 2a, we used the shape of an edge to indicate whether we are dealing with a downward or backward edge: Straight lines indicate downward edges, while curved lines denote upward edges (Fig. 8). This cue is very effective in graphs using iterative ranking. It provides a more natural cue, emphasizes cycles in the graph, and also prevents backward edges from being obscured because they are now shown outside the cluster structure. The effect is substantially less effective in graphs using cyclic ranking, though, in this case, we will have to fall back on using color.

4 Interactive Visualization

In the previous section, we described how the state transition graph can be visualized. However, this visualization is not the final answer yet. First, the amount of detail can be overwhelming, hence the user has to be enabled to focus and zoom in on various aspects. This can be easily dealt with. The user can select a certain cluster, upon which only this cluster and its descendants are shown, simplifying its inspection.

Second, only the structure of the state transition graph is displayed and not the information associated with the states and transitions. Typically, the relevant aspects of a system are modeled as a set of state variables. Each state is described by a particular valuation of these state variables, whereas a transition, occurring upon an event, leads to a change in the values of the state variables. Hence, the values of state parameters have to be shown to obtain more insight, especially in the relation of the structure of the graph with the semantics of the system studied.

One could aim for a single image, showing all information simultaneously, but we think this is not possible. In a large system, there can be tens to hundreds of state variables and their visualization for hundreds of thousands of states is a difficult multivariate visualization problem in

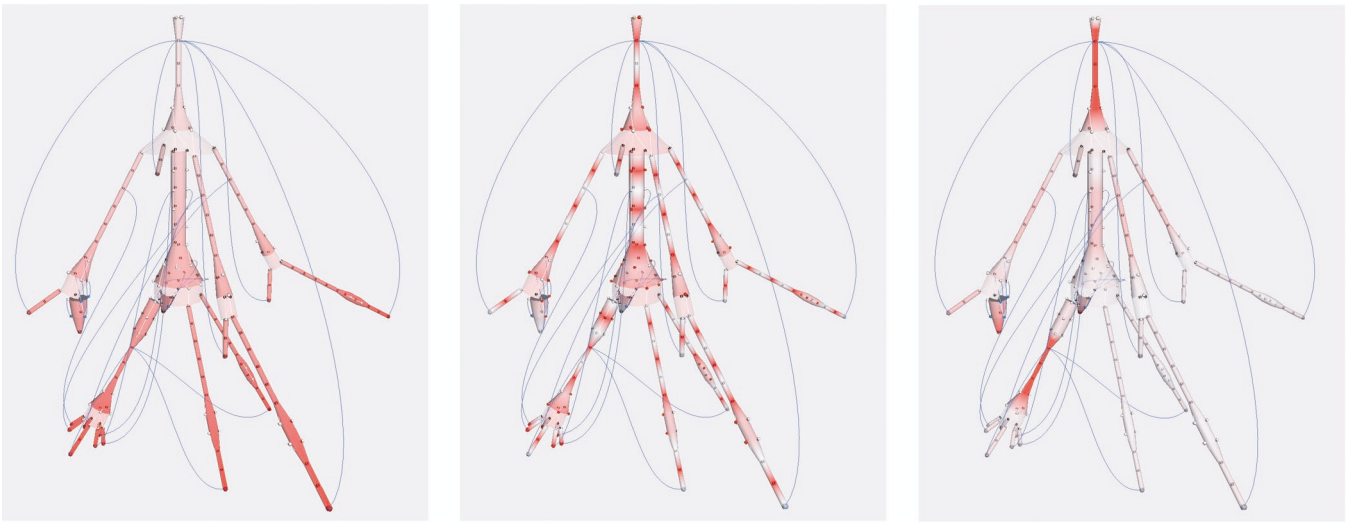

Fig. 9. Visualizing node attributes.

itself, even if we would not aim at relating them to the structure of the graph. Instead, we opt for interaction here. The user has to be enabled to ask questions on the system studied and has to be provided with visual answers.

Some examples of typical questions are:

- What is the value of a certain state variable?
- Which states are often visited?
- Given a start point, which states can be reached in a given number of steps?
- What is typical for a certain branch of the graph?

In this section, we present various solutions. In Section 4.1, we show how color can be used to visualize properties of clusters, in Section 4.2, we show how local structure can be emphasized, and, in Section 4.3, we present a correlation-based method to obtain information on sets of nodes.

4.1 Cluster Properties

Each node has associated attribute values. The display of these values per node is possible, for instance, via color. However, this will often give too much detail and, most of the time, only a fraction of all nodes will be visible. A convenient option is to visualize properties per cluster of nodes. We have realized this as follows: First, for each cluster, a certain property is calculated; second, the value of this property is mapped to a value between 0 and 1; and, finally, this value is used as an index in a predefined color table, which gives the color of the ring corresponding to a cluster. We implemented this using one-dimensional texture mapping such that we could use color interpolation in HSV space. As a result, the tree structure is smoothly colored according to the clusters' properties.

Several properties of clusters can be visualized. A simple property is the distance of a cluster from the root node. Visualization of this attribute enhances the notion of flow through the graph (Fig. 9a). It also helps in understanding the 3D structure of the tree: Overlapping transparent branches are now easier to distinguish.

Another important question of users is where nodes with some specific (combination of) attribute values are located. The corresponding property per cluster is the fraction of all nodes within the cluster that satisfy the request of the user (Fig. 9b). In our system, the user can select the state variable from a list and adjust its query value with a slider, thereby enabling him to brush quickly over a large range of values.

Stochastic information can be shown as well. Suppose that we start in the start node and make a random walk through the graph, what is the probability that our walk will end at a certain node? The probability can be estimated using the algorithm in Listing 2. This version is compact; a breadth first version, however, gives a better efficiency.

```
procedure Walk (n : Node; p : real);
begin
  if p < ε then
    n.Prob := n.Prob + p
  else
    q := p/N;
    n.Prob := n.Prob + q;
    forall m in next(n) do
    Walk(m, (p - q) / |next(n)| )
end;

forall n in V do n.Prob := 0;
Walk(StartNode, 1.0);
```

Listing 2. Probability algorithm.

We assume that the length of the walk is exponentially distributed with average length N. Equivalently, at each step of our walk, we have a chance of $1/N$ of ending the walk. At each step, we accumulate the probability q of ending the walk in the current state. Given that the probability of reaching a state equals p, q is equal to p/N.

Next, we make a step to all connected states and repeat this recursively. The probability of reaching a connected state is equal to $(p - q)/M$, with M the number of connected

states. Distribution of probabilities ends when the probability p of reaching a state falls below a certain threshold ε.

Normalizing the resulting probability distribution to [0..1] again yields a lookup color value, which can be used to color the cluster structure. Fig. 9c shows a sample probability distribution. Areas located immediately after backpointers light up because they are relatively highly traveled. This process also spends relatively much time in the small "sink" in the middle left of the picture. Several variations can be imagined. For instance, more realistic results can be obtained if, for each type of transition, the probability is known. In that way, less visited (i.e., exception handling) routines would show up less brightly colored.

In a similar manner, many more options for attribute visualization are possible: We could choose to display the average fan out per cluster, which highlights the more complex sections of a graph, or the average change in state values, highlighting sections where the values of state variables change abruptly.

4.2 Node Details

Although the overview provides important structural information, in some cases, it might be desirable to view the actual connections between nodes in detail. Execution might end up in a particular branch of the structure because of choices made much earlier in execution. When dealing with a visualization displaying a large number of nodes and edges at once, such causalities are impossible to trace.

We solved this by allowing the user to select a node, after which the system also selects nodes that can be reached in maximally N steps. This selection of the neighborhood of a node allows the user to immediately see the effects of choosing a particular path on the choices to be made later on. In the same way, it is possible to select a node and have the system also select nodes from which the selected node can be reached in maximally N steps. Since we wish to be able to use color to display additional node properties, we increase the size of a node to indicate it is selected. This has the pleasant side effect of visually emphasizing the node structure. To enhance this effect, selected edges (edges with both of their nodes selected) are drawn thicker, while the transparency of the backbone structure is increased.

Fig. 10 shows a part of a sample structure in which we selected one of the uppermost states. It can clearly be seen that execution continues into either the left or center branch. Subsequent analysis shows that, depending on which initial state is selected, execution progresses into either one of the outside branches, while the center branch is accessible from all states.

4.3 Correlating Attributes and Locations

An observer will usually also want to know what the typical characteristics of particular clusters are. If he can identify a single common value of a state variable for the particular branches in the system, he can pin higher-level semantic concepts on different areas of the graph by using his knowledge on the state variables. We have integrated this into our application by allowing the user to select a substructure and let the system determine the correlation between a set of nodes with a common property and nodes

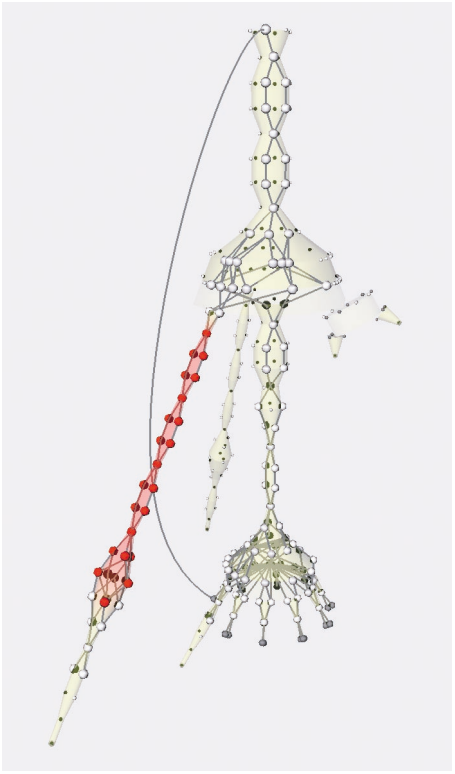

Fig. 10. Tracing node paths.

in the selected region. The correlation between the two properties x and y over N samples can be given by the Pearson correlation coefficient r:

$$r = \frac{N\Sigma xy - \Sigma x \Sigma y}{\sqrt{(N\Sigma x^2 - (\Sigma x)^2) \cdot (N\Sigma y^2 - (\Sigma y)^2)}}.$$

Suppose we wish to know the correlation between the two properties "*a node has value v for state parameter p_i*" and "*a node is an element of a selected set of nodes S.*" If we substitute a numeric value of 1 for true and 0 for false, we obtain a binary value pair for each of the N nodes. The correlation coefficient can then be computed by substituting:

$$\begin{aligned}\Sigma x^2 &= \Sigma x = |\{n \in V : n.p_i = v\}| \\ \Sigma y^2 &= \Sigma y = |S| \\ \Sigma xy &= |\{n \in V : n.p_i = v\} \cap S|.\end{aligned}$$

Computing r for all possible combinations of p_i and v yields a list of correlation factors between -1 and 1. A value close to 1 indicates that a property $p_i = v$ is typical for the selected region. A value of -1 indicates that none of the nodes in S have the property $p_i = v$, while all other nodes do.

5 Cases

This section presents a number of real world cases in which we applied this visualization method to analyze the state space.

5.1 Modular Jack System

In this first case, we analyzed a modular hydraulic jack system. If needed, extra hydraulic jacks can be added to an

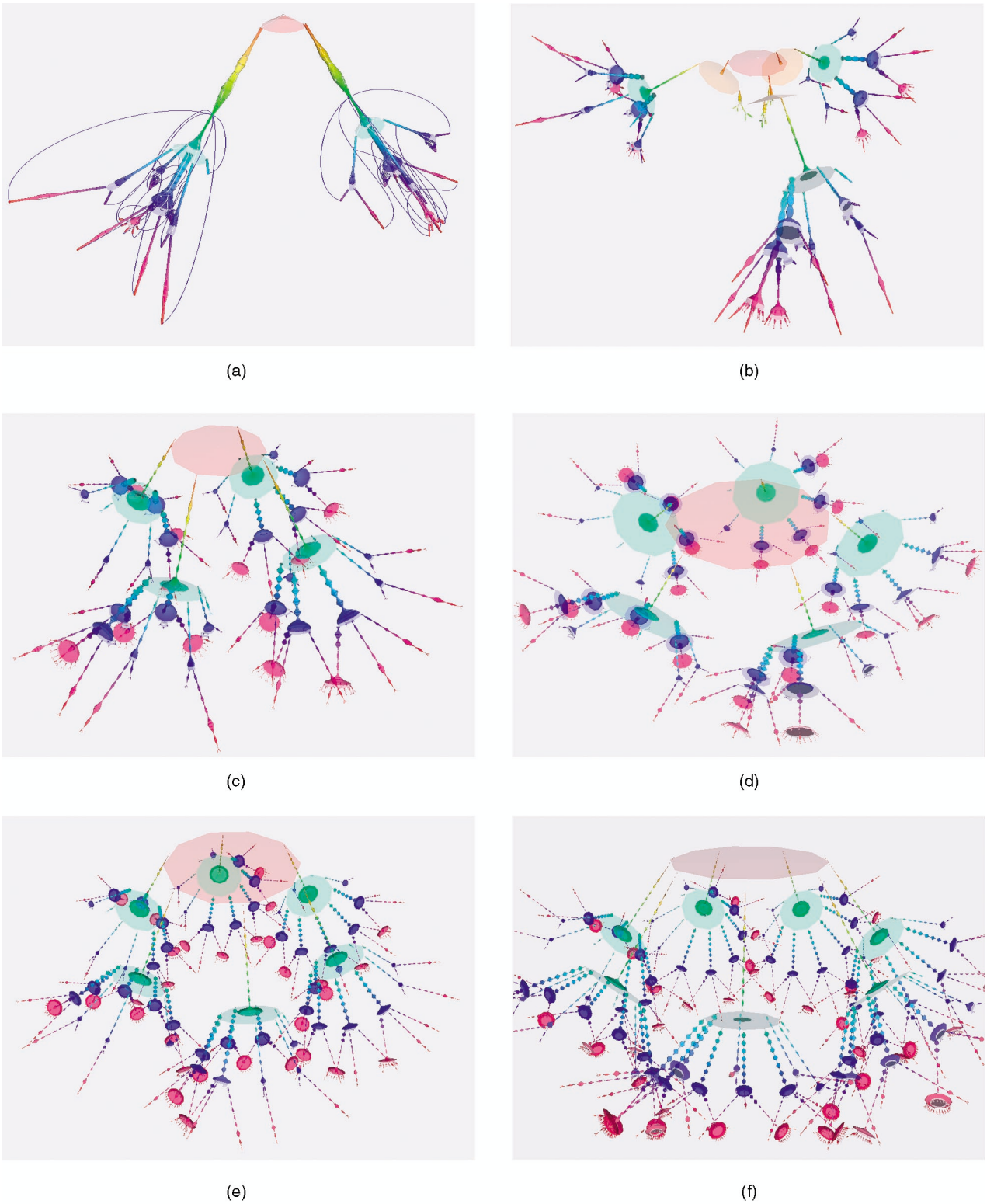

Fig. 11. Behavior of modular jack system consisting of (a) 512 nodes, (b) 2,860 nodes, (c) 15,609 nodes, (d) 70,926 nodes, (e) 284,077 nodes, and (f) 1,025,844 nodes.

existing setup of similar jacks to increase maximum lift capacity. The whole setup of jacks can be operated from the controls of any individual jack, so, obviously, some kind of synchronization will be necessary. The corresponding synchronization protocol was developed at Weissmuller and was subsequently analyzed by the CWI in Amsterdam [5]. Fig. 11 shows the behavior of the protocol visualized using iterative ranking, with the number of jacks increasing from two in the first picture up to seven in the last picture. Although there are major differences in complexity between the protocols, the general behavior turns out to be very similar. This figure also shows the scalability of our

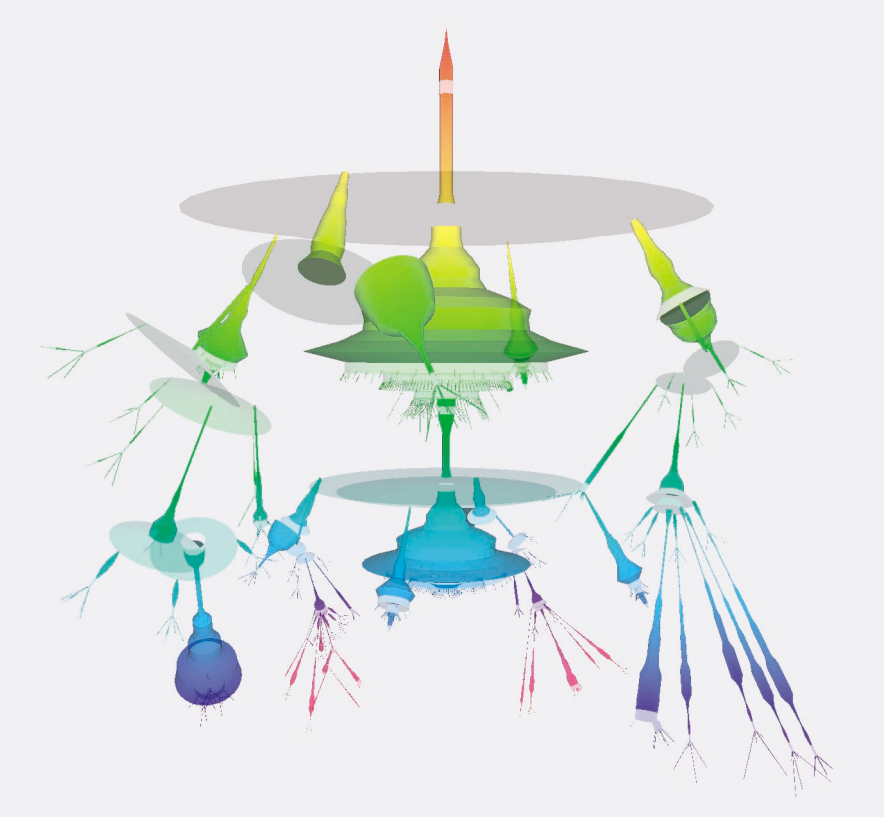

Fig. 12. Behavior of the link layer of the Fire Wire protocol, simulated using two links (25,898 nodes).

visualization method: from about 500 nodes in the first picture up to over one million nodes in the last picture. Important features of all behaviors are the relatively thin strings of clusters that branch off at the top of the visualization. Fig. 11a clearly shows that it is impossible to return to the start node after commencing execution since there are no backpointers returning to the start node. This means that these clusters must belong to an initialization phase. Another striking feature is the perfect symmetry between each leg in the protocol. This can be explained by the fact that it must be possible to operate the entire setup from any one lift, so the protocol has to behave identically for each controlling jack.

5.2 IEEE 1394—Fire Wire

A second case we applied this visualization method to was the Fire Wire protocol [7]. Fire Wire is a high-speed serial protocol which is currently widely used in digital video processing. We confined ourselves to analyzing the link layer of the protocol, which provides an interface between the high-level transaction layer and the physical layer. A formal description [10] of the protocol was used to simulate the behavior of two Fire Wire links connected via a serial bus.

One feature we focused on is the two largely similar bulky areas in the center of the visualization (Fig. 12). As a start point for analysis, we selected the bottom part and the system correlated this part with one state value that was unique for that part. Looking at the original specification, we determined that this variable represented a Boolean array of length 2 that kept track of which links have requested control over the bus during a so-called *fairness interval* and had a value of [true, true] in the selected region. During each fairness interval, each link may request control over the bus at most once. A fairness interval ends when the bus has been idle for a specific amount of time, after which the system returns to its initial state. In the top part, either one of the two links gains access to the bus for communication. If the bus does not stay idle for a specific amount of time, that is, a link requests fair use of the bus, the system grants this use only if the link has not used the bus before. After that, the system exhibits similar communication behavior as in the top part.

6 Conclusion and Recommendations

We have presented a new method for the visualization of state transition graphs. Instead of computationally optimizing one aesthetic, we chose a more procedurally oriented approach, focusing on structure symmetry. The resulting visualizations give the user an overview of the entire graph and the ability to view the detailed node structure if desired. The strong focus on symmetry and the predictability of this method allow users to compare graphs that are similar in overall structure, but have different local properties, making it potentially suitable for other types of directed graphs. The proposed interactive options allow quick access to detail information on demand. Although this method performs well for large graphs (up to approximately one million nodes) with fairly low connectivity, clusters in highly connected graphs tend to become too large, resulting in a less effective overview. Performance of the method is good; using a PIII 1 GHz, we were able to

cluster a graph of 284,077 nodes (Fig. 11e) in 36 seconds, 20 of which were spent loading the raw graph data from disk. After this preprocessing stage, images could be generated interactively. Memory requirements are approximately 1 KB per node, preventing us from accurately measuring the performance on a graph of one million nodes. From the viewpoint of state space analysis, further work should focus on integrating existing tools and methods for graph reduction into the visualization. On the visualization side, problems still remain in the display of edges and nodes, which still tends to become too cluttered. As a potential solution, one can think of recursively applying this method to more complex sections of the graph, which may split larger clusters into smaller ones, or adding extra depth or motion cues to edges in the graph. A 2D version of this method may also be considered. Though requiring more visualization space, a two-dimensional layout may provide better detail views, especially for graphs of higher connectivity. An equally important, but somewhat problematic area here is user testing. Since almost no insight into large state spaces exists, interpretation requires highly skilled experts and sometimes raises more questions than it answers. We observed that all experts were excited, intrigued, and sometimes also puzzled by the new perspective on state spaces offered by this visualization method.

ACKNOWLEDGMENTS

Thanks go to Jan Friso Groote and Judi Romein of the Systems Engineering Group at the Technische Universiteit Eindhoven for their expertise and additional feedback. The authors would also like to thank the Center for Mathematics and Computer Science (CWI) in Amsterdam. This work was supported by the Netherlands Organisation for Scientific Research (NWO) under Grant 612.000.101.

REFERENCES

[1] A. Arnold, *Finite Transition Systems.* Prentice Hall, 1994.
[2] G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.
[3] J. Carrière and R. Kazman, "Research Report: Interacting with Huge Hierarchies: Beyond Cone Trees," *Proc. IEEE Conf. Information Visualization '95,* pp. 74-81, 1995.
[4] P. Eades and Q.W. Feng, "Multilevel Visualization of Clustered Graphs," *Proc. Graph Drawing '96,* pp. 101-112, 1996.
[5] J.F. Groote, J. Pang, and A.G. Wouters, "A Balancing Act: Analyzing a Distributed Lift System," technical report, Dept. of Software Eng., CWI, Amsterdam, to appear.
[6] F. van Ham, H. van de Wetering, and J.J. van Wijk, "Visualization of State Transition Graphs," *Proc. IEEE Conf. Information Visualization '01,* pp. 59-66, 2001.
[7] IEEE Computer Soc., "IEEE Standard for a High Performance Serial Bus," Standard 1394-1995, Aug. 1996.
[8] R.J. Hendley et al., "Narcissus: Visualising Information," *Proc. IEEE Symp. Information Visualization,* pp. 90-96, 1995.
[9] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data.* Prentice Hall, 1988.
[10] S.P. Luttik, "Description and Formal Specification of the Link Layer of P1394," Technical Report SEN-R9706, CWI Amsterdam, 1997.
[11] T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," *Proc. Virtual Reality Modeling Language (VRML '95) Symp.,* 1995.
[12] G.G. Robertson, J.D. Mackinlay, and S.K. Card, "Cone Trees: Animated 3D Visualizations of Hierarchical Information," *Human Factors in Computing Systems, CHI '91 Conf. Proc.,* pp. 189-194, 1991.
[13] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical Systems Structures," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 11, no. 2, pp. 109-125, 1981.
[14] W. Tutte, "How to Draw a Graph," *Proc. London Math. Soc.,* vol. 3, no. 13, pp. 743-768, 1963.
[15] C. Ware et al., "Layout for Visualizing Large Software Structures in 3D," *Proc. VISUAL '97,* pp. 215-223, 1997.

Frank van Ham obtained the MSc degree in computer science (with honors) from the Technische Universiteit Eindhoven in 2000. He started as a PhD student at the same university in 2001. His current research is focused on the development of new techniques for the visualization of large graphs. Among his primary research interests are information visualization, computer graphics, and AI.

Huub van de Wetering studied mathematics at the Technische Universiteit Eindhoven. He received the PhD degree from the same university, studying the representation of discrete curves with chain codes. He is currently an assistant professor in computer graphics and researches in the areas of computer animation, implicit surfaces, VRML, and information visualization.

Jarke J. van Wijk received the MSc degree in industrial design engineering in 1982 and the PhD degree in computer science in 1986 from Delft University of Technology, both with honors. He is a full professor in visualization at the Technische Universiteit Eindhoven. He has (co)authored about 40 publications on visualization. His main research interests are information visualization and flow visualization, both with a focus on the development of new visual representations.