Visualization of State Transition Graphs

Frank van Ham, Huub van de Wetering, Jarke J. van Wijk
Eindhoven University of Technology
Dept. of Mathematics and Computer Science
P.O.Box 513, 5600 MB Eindhoven, The Netherlands
{fvham, wstahw, vanwijk}@win.tue.nl

Abstract

A new method for the visualization of state transition graphs 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. The resulting visualization makes it easier to relate features in the visualization of the state transition graph to semantic concepts in the corresponding process and vice versa.

1. Introduction

Graphs play an important role in science with applications ranging from biology to electronic engineering. This paper focuses on an application in computer science, where graphs are applied to describe Finite State Machines [1]. The concept of a finite state machine plays a central role in computer science, as it can be used to model processes. Subsequent study of these models can then reveal useful information about the process. Although some aspects, such as deadlock states, can be identified by computational analysis, properties like symmetries within or similarities between processes can be nearly impossible for a computer to find. Since the human perceptual system is much more adept at finding similarities between structures a good visualization is indispensable. Such a visualization is trivial when dealing with a small number of states, since a simple node and link diagram can be used. In practice however, the finite state machines under analysis often consist of thousands of nodes, which severely limits the usefulness of the traditional method (Fig 1). This paper presents a new visualization method that maintains symmetry present in the graph, while providing the user with an overview of the entire graph's global structure. For a quick impression, figures 10 and 11 show visualizations of graphs with 191 and 2860 nodes, respectively.

Section 2 discusses related work and the general ideas behind this type of visualization. Section 3 describes the method used to construct it in more detail. Results of applying this method to a number of sample cases are presented in section 4. Finally, section 5 summarizes these results and offers some recommendations for improvements.

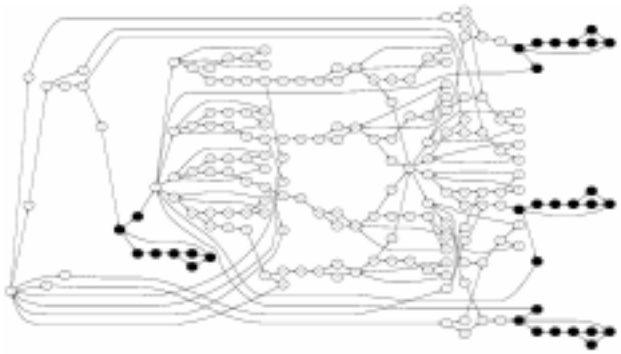

Fig 1. 2D Visualization of a state transition graph with 191 states

2. Approach

Much research has been done in the area of two dimensional (2D) graph drawing. For an overview see for instance [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 different techniques such as hyperbolic space [9], hierarchical node clustering [13],[4] or self organizing

networks [7]. 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 symmetry and regularity. The visualization method presented in this paper relies on two 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 figure 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 [8]. Clustering based on the structure of the graph is especially useful here, since the structure 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 closer.

3. Visualization

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 individual nodes and edges.

3.1 Ranking

In a first step nodes are assigned to a layer or rank, comparable to the standard Sugiyama-type layout [11]. That is, all nodes are assigned a positive discrete value that indicates their depth relative to the start node. We found two ranking methods the most useful, since they correspond with two types of views on processes: iterative and cyclic (Fig 2).

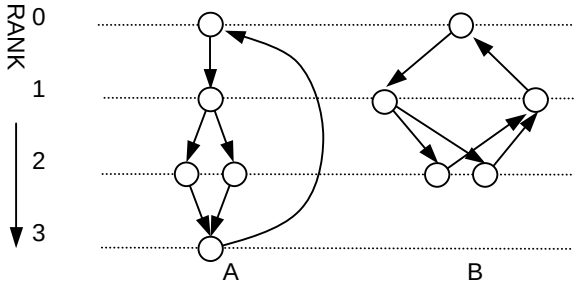

Fig 2. Different views on the same process: Iterative (A) and Cyclic (B)

In an iterative process view, a start node s is assigned a rank of 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.

We aim at the creation of a tree structure based on the original graph which we then later 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 positive 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 $(x=z_1,z_2,...,z_N=y)$ of nodes with equal rank exists, such that for all $1 \leq i < N$ $D(z_i) \cap D(z_{i+1})$ is not empty (see Fig 3).

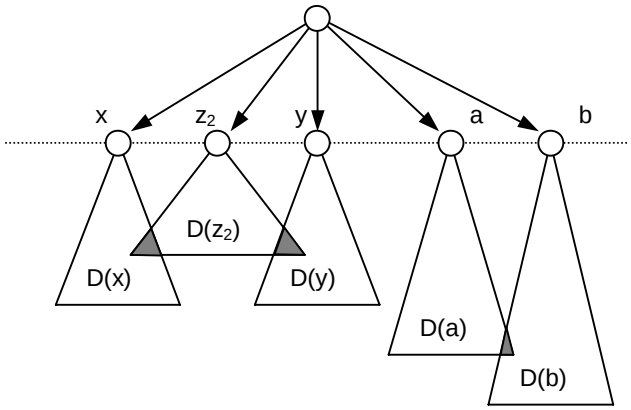

Fig 3. Nodes x and y are equivalent, as are nodes a and b, while nodes x and b are not.

It can be fairly easily shown [6] that this is an equivalence relation, so by definition its equivalence classes are non-empty 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.

The clusters defined in this section together with their ancestor relations have a number of interesting properties:

- Every node is in exactly one cluster;
- Cyclic ancestor relations are not possible since an ancestor of a cluster C has a rank that is one lower than the rank of C;
- Each cluster has at most one ancestor.

Based on these properties we can state that the resulting structure is a tree structure. Consequently, we can use or modify existing visualization methods for trees to visualize the clustered structure. The next section discusses this step in more detail.

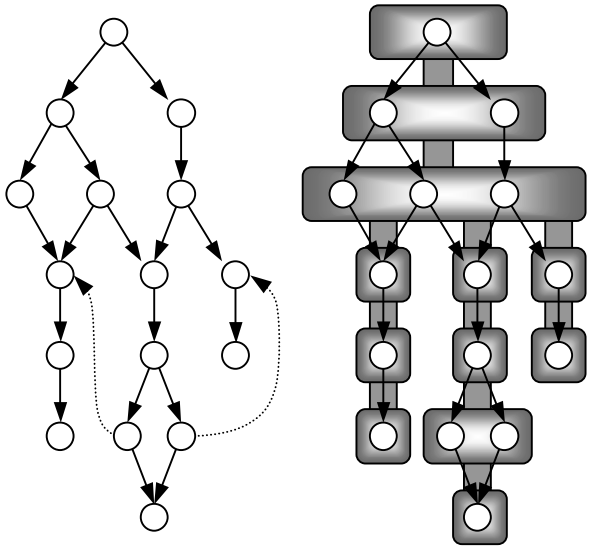

Fig 4. Original graph G (left) and clustered graph G' with backbone tree (right)

3.3 Visualizing the backbone tree

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

1. **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.
2. **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.
3. 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 [10], 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 status as tree nodes in a cone tree. To emphasize the hierarchy in the cluster structure, truncated cones are drawn between related clusters. Figure 5 gives a quick impression. The overall process adheres to the basic concepts of cone trees but with a few alterations:

A. Clusters (the nodes in the cone tree) are visualized as circles of different sizes.
B. Symmetry is improved by also allowing clusters to be positioned in the center of the cone's base.
C. 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 our 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.

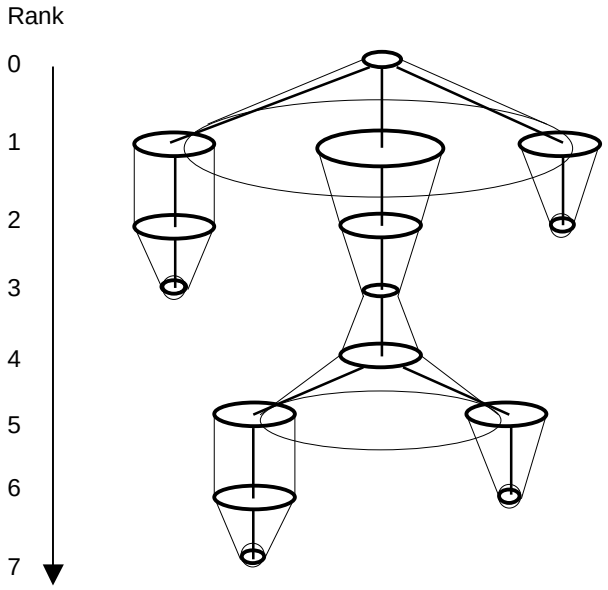

Fig 5. Sample visualization

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.

- N = 1: In this case the descendant cluster is positioned directly below A.
- 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 be 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 6A).
  - If there is one unique smallest cluster among the descendant clusters we center this cluster when there are no largest clusters centered (Fig 6B) 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 non-centered cluster remains, we choose not to center the largest cluster. This produces a more balanced layout (Fig 6C).

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 non-centered 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. Fig 10 shows some sample visualizations created with a prototype program.

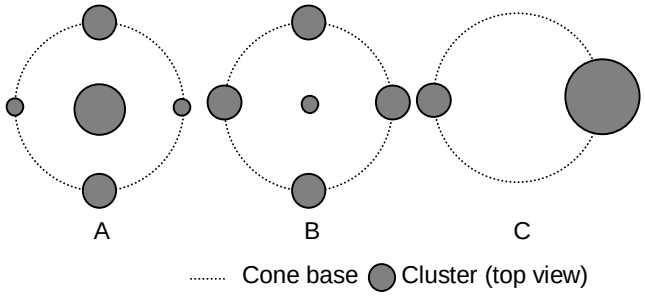

Fig 6. Layout of individual clusters

3.4 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 parentnode 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 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 and descendant clusters, similar to [12]. In a second step we adjust these initial positions of nodes to increase the space between them. The following describes these two steps in more detail.

Step 1: The position of a node is based on the positions of connected nodes. An initial positioning is made in a bottom-up pass (i.e. nodes with higher ranks are processed first, so node positions depend on the position of connected nodes in descendant clusters). This pass is followed by a top-down and another bottom-up pass where we resolve the positions of nodes that could not be placed in the first pass.

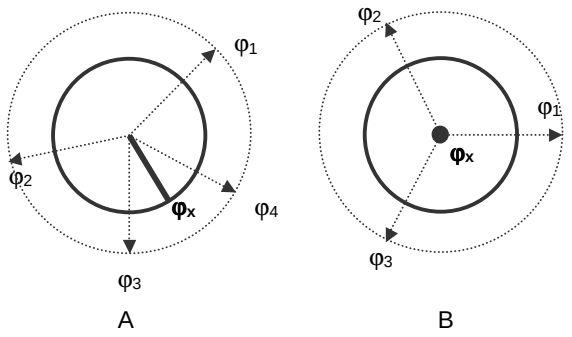

Fig 7. Calculating node positions (top view)

Suppose a node x has N children positioned at angles φ_1, φ_2,…, φ_N. We can then calculate an optimum position for x based on the average positions of the nodes in a descendant family. Let this position be (r, φ_x) in polar coordinates. If r is below a threshold τ we choose to position node x in the center of the cluster, otherwise we position x on the circle edge at angle φ_x.

Using this method we can calculate a position φ_x for an individual node x in a cluster C using the following rules:

- **|C| = 1**: If node x is the only node in C we center it.
- **|C| >1**: Assuming node x has N childnodes and cluster C has D descendants, we discern between:
  - **D = 1**: Calculate φ_x as outlined above.
  - **D >1**: For all N childnodes in the different clusters calculate angles $\varphi_1, \varphi_2, \ldots, \varphi_N$ relative to cluster C and calculate φ_x as outlined above.
  - **D = 0** or **All N childnodes are centered and D=1**: In this case there is not enough structural information available to position the node. Nodes that cannot be assigned a position in the first bottom-up pass are temporarily assigned to the center position and marked as undecidable. In the second pass we use exactly the same heuristic but apply it top-down to the undecidable nodes, basing their position on the position of their parentnodes in the cluster's ancestor. Nodes for which a position still can't be uniquely determined are positioned in a third bottom-up pass, in which case we make a deterministic decision for a position based on maximizing the distance between nodes in a cluster (Fig 8).

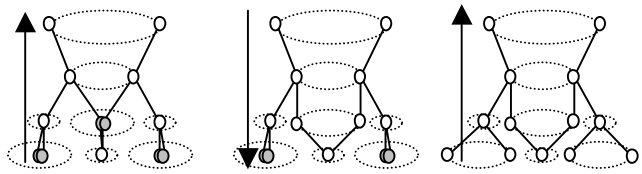

Fig 8. Three pass heuristic. Nodes marked as undecidable after a pass are colored gray.

Step 2: After positioning nodes in a cluster based on the above heuristic we increase space between them in a second step. We wish to:

- Leave the (partial) order on nodes we created in the previous step intact.
- Distribute the nodes in a balanced and symmetrical manner over their childnode(s)

The following algorithm outline presents a simple but effective way to accomplish this:

Fix a number of equidistant positions, called slots, on the circle boundary; (Fig 9A)
Round each node position to the nearest slot; (Fig 9B)
for** each slot S **do
if** S contains more than one node **then
Calculate the maximum circle section available for that slot, considering the occupation of neighboring slots;
Distribute nodes in a balanced manner over that section, conserving the order created in step 1; (Fig 9C)

In the sample below we used 4 slots for tutorial purposes. For a detailed description of the algorithm including how to determine an optimal number of slots and how to distribute multiple centered nodes, we refer to [6].

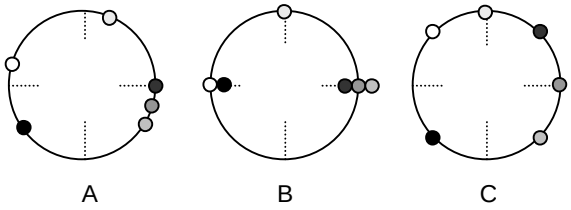

Fig 9. Distributing individual nodes

This method of positioning nodes gives acceptable and predictable results for graphs of low to medium connectivity and maintains $O(|V|)$ performance. Finite state machines tend to be large and have a low connectivity, which makes the above method well suited for our purpose.

4. Results

We implemented the above visualisation method in a prototype program using OpenGL in a MS Windows environment. A number of options for interaction and viewing were also implemented. It was tested with graphs up to approximately 25.000 nodes and proved especially useful in two cases that we will discuss below.

4.1 Searching for clusters with a common property

This section discusses the application of this method to two real world cases. Figure 10 shows visualizations of the same graph as displayed in figure 1, describing the Alternating Bit Protocol. The most striking feature in the cyclic visualization (10A) are the two lobes at the bottom. Since a cyclic ranking has no backpointers and all other edges fall within the outline given here, transitions from one lobe to another are only possible using states in the top part. The iterative visualization (10B) clearly shows the 4-fold symmetry in the protocol but the (marked) nodes in the left lobe in figure 10A are spread out over the visualization due to the long backpointers. Although cyclic ranking does not convey the flow in a graph as well as iterative ranking, it keeps every node close to its neighbors making it a useful tool to locate clusters of related nodes. An observer will usually also want to know what the typical characteristics of such clusters are. If he can identify a single common value of a state variable for the nodes in a lobe in figure 10A, he can pin higher-level semantic concepts on different areas of the graph by using knowledge of the state variables. We have integrated this into our prototype 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

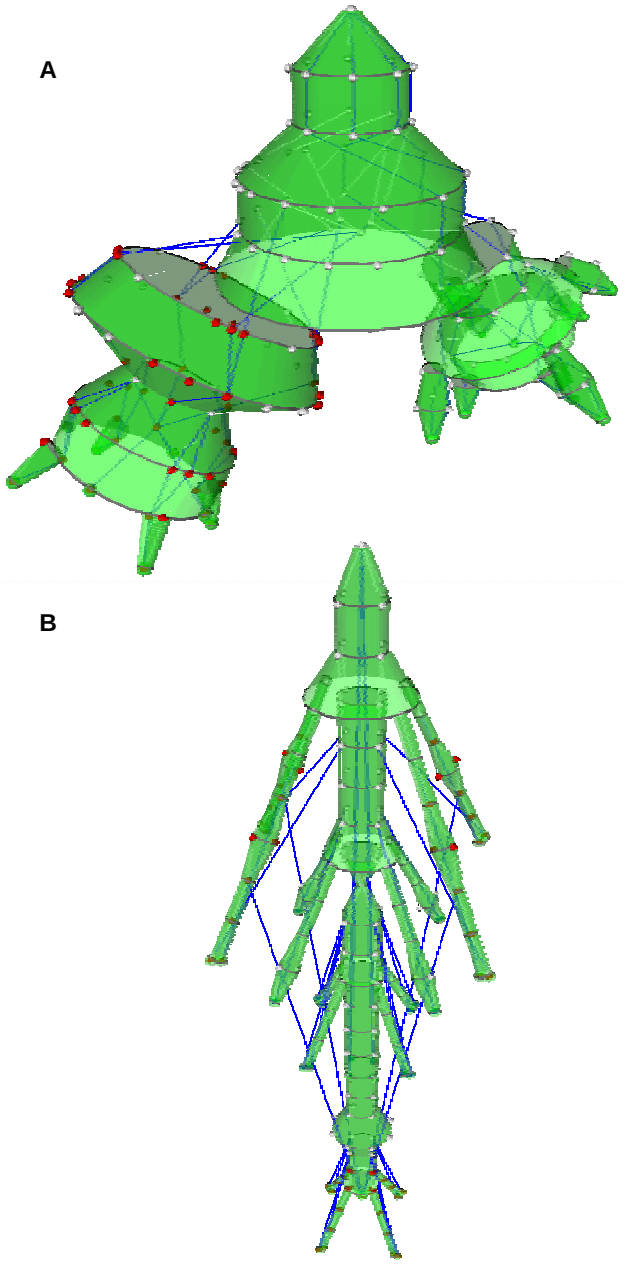

Fig 10. Alternating Bit Protocol (191 nodes) with
A) Cyclic ranking
B) Iterative ranking with backpointers shown

in a selected region.

For each possible state variable / value combination the Pearson correlation coefficient is calculated. This gives us a list of correlation factors of which the ones closest to 1 indicate which common property is present in the nodes in the selected region.

4.2 Identifying symmetries and similarities

A clear picture of the symmetry in a graph can reduce the amount of time needed to analyze the structure, since one can suffice with the analysis of only one of the symmetrical parts. A second advantage is that symmetrical features help the user break down an initially complex structure into more digestible chunks. Although one could argue that symmetrical features could also be

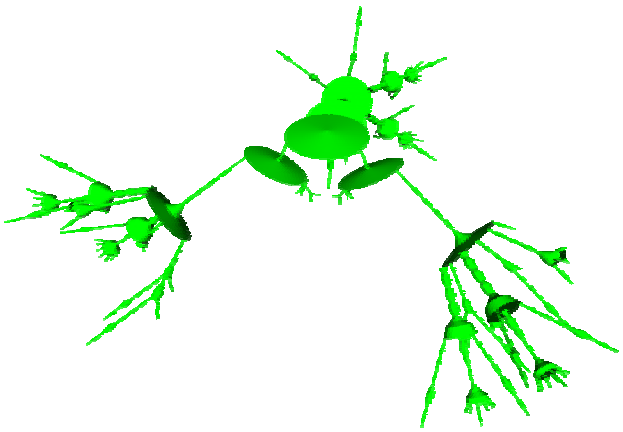

Fig 11. Protocol for a 3-jack car lift system with 2860 nodes and iterative ranking

extracted by a computer searching all nodes of the graph, the same is not the case for similar features. In this case the skills of the human perceptual system with respect to pattern recognition are indispensable. Figure 11 shows a graph consisting of 2860 nodes, depicting the communication protocol for a modular car lift system [5] where three hydraulic jacks support one platform. If we compare the visualization of this graph to the one depicting the same protocol for two jacks we can expect to find some similarities, since both deal with the same real-world process. Figure 12 shows a section in the bottom right of figure 11 next to a part of the 2-jack protocol. Both sections clearly have similar global structures, but a more complicated detail structure in the case of three jacks. This is also the case for many other sections. This allows us to apply concepts obtained from analysis of the simple version to the more complicated version of the protocol. Such similarities are nearly impossible for a computer to find, and it is doubtful that even a user using the standard 2-dimensional visualization would have picked them up, which makes this visualization method very useful in this respect.

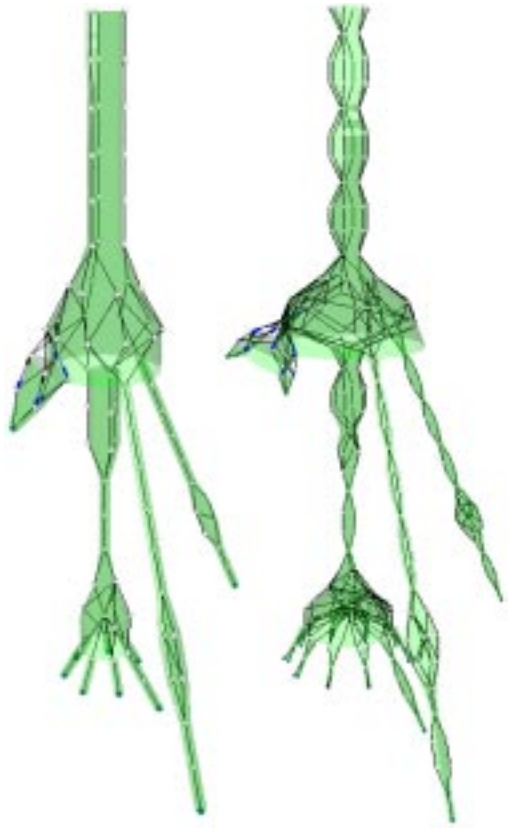

Fig 12. Comparing a section in figure 11 to a section of the protocol for a 2-jack car lift

5. 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 also suitable for other types of directed graphs. Although this method performs well for large graphs with fairly low connectivity, clusters in highly connected graphs tend to become too large, resulting in a less effective overview. Comprehending the actual connections between nodes when looking at a 2D projection of a 3D scene also proves difficult.

Further work should focus on methods to alleviate these problems. 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.

Acknowledgements

We would like to thank Jan Friso Groote and Michel Reniers of the systems engineering group at the Eindhoven University of Technology for their expertise and additional feedback.

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 Visualisation of Graphs”, *Prentice Hall, 1999.*

[3] J. Carrière and R. Kazman, “Research Report: Interacting with Huge Hierarchies: Beyond Cone Trees”, *Proceedings of the IEEE Conference on Information Visualization ’95*, IEEE CS Press, pp. 74-81, 1995.

[4] P. Eades and Q.W. Feng, “Multilevel Visualization of Clustered Graphs”, Graph Drawing 96, *Springer Lecture notes in Computer Science 1190*, pp. 101-112, 1996.

[5] J.F. Groote, J. Pang and A.G. Wouters, “A Balancing Act: Analyzing a Distributed Lift System”, to appear as a technical report of the Department of Software Engineering, CWI, Amsterdam.

[6] F. van Ham, “Visualization of State Transition Graphs”, Master’s thesis available from www.win.tue.nl/~vanwijk

[7] R. J. Hendley et al., "Narcissus: Visualising Information", *Proceedings of the IEEE Symposium on Information Visualization*, IEEE CS Press, pp. 90-96, 1995.

[8] A.K. Jain and R.C. Dubes, “Algorithms for Clustering Data”, *Prentice Hall*, 1988.

[9] T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space", *Proceedings of the VRML'95 Symposium,* ACM SIGGRAPH, ACM Press, 1995.

[10] 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 Conference Proceedings, ACM Press*, pp. 189-194, 1991.

[11] K. Sugiyama, S. Tagawa and M. Toda, “Methods for Visual Understanding of Hierarchical Systems Structures”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11 No. 2, pp 109-125, 1981.

[12] W. Tutte, “How to draw a graph”, *Proceedings of the London Mathematical Society* Vol. 3 No. 13, pp. 743-768, 1963.

[13] C. Ware et al., “Layout for visualizing Large Software Structures in 3D”, *Proceedings of VISUAL'97*, pp. 215-223.